Programming languages — Ada

DEFECT REPORTS

Part 3 (Draft 4)

For ISO/IEC 8652:2012

October 2015

This document was prepared by AXE Consultants.

Introduction

This document contains defect reports on the Ada 2012 standard [ISO/IEC 8652:2012], and responses formulated by the Ada Rapporteur Group (ARG) of ISO/IEC JTC 1/SC 22/WG 9, the Ada working group. The ARG is the language maintenance subgroup of WG 9, meaning that it is responsible for determining the corrections to the standard.

Defect Reports usually come from comments submitted by the public to the ARG. These comments are distilled into a question, given in the **Question** section of the Defect Report response.

In order to formulate the response to the Defect Report, the question is carefully considered and often discussed at length by the ARG. The results are recorded in the **Discussion** section of the response. The answer to the question arrived at after the discussions is summarized in the **Summary of Response** section. A more detailed answer to the question can be found in the **Response** section. Sometimes, the issue is so obvious that there is no **Response** or **Discussion** section. If the result of the discussion finds that some change to the standard would be required to arrive at an answer to the question, a **Corrigendum Wording** section includes the specific wording change to standard. These **Corrigendum Wording** sections are gathered together in a Technical Corrigendum or Amendment document.

A Defect Report and Response is the final step of a lengthy process of formulation, discussion, and approval. The working documents of the ARG (called Ada Issues) contain additional information about the issue and its resolution. Ada Issues may include sections for testing information (**ACATS test**), informal wording changes (**Wording**), ASIS changes needed (), and an appendix including E-Mail comments on this issue (**Appendix**). These sections are not included in the Defect Reports found in this document. This information is available in the Ada Issues documents, which can be accessed on the web at www.ada-auth.org/arg.

The Defect Reports and Responses contain many references of the form ss.cc(pp) or ss.cc.aa(pp). These refer to particular paragraphs in the standard, with the notation referencing the (sub)clause number in the Ada 95 standard (ss.cc.aa), followed by a parenthesized paragraph number (pp). Paragraphs are numbered by counting from the top of the (sub)clause, ignoring headings.

The Defect Reports and Responses contain references to the Annotated Ada Reference Manual (AARM). This document contains all of the text in the Ada 2012 standard along with various annotations. It was originally prepared by the Ada 95 design team, and has been updated by the ARG for later versions of Ada. It is intended primarily for compiler writers, test writers, and the ARG. The annotations include rationale for some rules. The AARM is often used by the ARG to determine the intent of the language designers.

The Defect Reports and Responses may contain references to Ada 95. Ada 95 is the common name for the previous version of the Ada standard, ISO/IEC 8652:1995. AI95 refers to interpretations of that standard. Similarly, Ada 83 is the common name for the original version of the Ada standard, ISO/IEC 8652:1987. AI83 refers to interpretations of that standard.

This document contains all of the Defect Reports used to prepare Ada Technical Corrigendum 1 for Ada 2012. Resolutions of newer issues can be found on the web site mentioned previously.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0117**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **0.2 0.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0117 Add Raise Expression to Introduction

Working Reference Number AI12-0141-1

Question

The Introduction, paragraph 57.16/3 describes the kinds of expressions added by Ada 2012. AI12-0022-1 added raise expressions, which is another kind of expression associated with Ada 2012. Should it be mentioned in the Introduction? (Yes.)

Summary of Response

Add interfaces to the list of types that can have invariants in paragraph 57.15/3 of the introduction. Add raise expressions to the list in paragraph 57.16/3 of the introduction.

Corrigendum Wording

Replace 0.2(57.15):

- The concept of assertions introduced in the 2005 edition is extended with the ability to specify preconditions and postconditions for subprograms, and invariants for private types. The concept of constraints in defining subtypes is supplemented with subtype predicates that enable subsets to be specified other than as simple ranges. These properties are all indicated using aspect specifications. See subclauses 3.2.4, 6.1.1, and 7.3.2.

by:

- The concept of assertions introduced in the 2005 edition is extended with the ability to specify preconditions and postconditions for subprograms, and invariants for private types and interfaces. The concept of constraints in defining subtypes is supplemented with subtype predicates that enable subsets to be specified other than as simple ranges. These properties are all indicated using aspect specifications. See subclauses 3.2.4, 6.1.1, and 7.3.2.

Replace 0.2(57.16):

- New forms of expressions are introduced. These are if expressions, case expressions, quantified expressions, and expression functions. As well as being useful for programming in general by avoiding the introduction of unnecessary assignments, they are especially valuable in conditions and invariants since they avoid the need to introduce auxiliary functions. See subclauses 4.5.7, 4.5.8, and 6.8. Membership tests are also made more flexible. See subclauses 4.4 and 4.5.2.

by:

- New forms of expressions are introduced. These are if expressions, case expressions, quantified expressions, expression functions, and raise expressions. As well as being useful for programming in general by avoiding the introduction of unnecessary assignments, they are especially valuable in conditions and invariants since they avoid the need to introduce auxiliary functions. See subclauses 4.5.7, 4.5.8, 6.8, and 11.3. Membership tests are also made more flexible. See subclauses 4.4 and 4.5.2.

Discussion

We also add "interfaces" to "private types" as types that can have an invariant.

None of the other changes of the corrigendum seem worth mentioning in the Introduction.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0118**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **1.1; 1.1.2; 3.9; 3.10; 5.2; 6.1; 6.2; A.18.25; A.18.26**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0118 Presentation errors in Ada 2012

Working Reference Number AI12-0056-1

Question

1) The Next_Frame example in 6.1(39) uses type Frame, which is labeled "see 3.10". But there is no type Frame defined in 3.10. Should there be? (Yes.)

2) Ada 2012 allows functions to have "in out" and "out" parameters, but the examples in 6.1 do not show any such function declarations. Should there be one? (Yes.)

3) If one compiles the second assignment in the example of 5.2(20), my compiler reports "type of aggregate cannot be class-wide". This appears to be correct, the example is wrong as written. Should it be fixed? (Yes.)

4) 1.1(3/3) has an extra "and" preceding "random number generation". Should it be deleted? (Yes.)

5) 1.1.2(24/3) starts "Each section is divided into subclauses...", but of course it is a "clause" that is divided into subclauses. Should this be fixed? (Yes.)

6) In 3.9(12.4/3), "derived_type_declaration" clearly refers to a syntax term, but it is not in the syntax font. Should this be fixed? (Yes.)

7) The function Copy declared in A.18.25(10/3) takes a Tree parameter, but returns a List! There is no List declared in this package; this ought to be Tree, right? (Yes.)

8) Formal procedure Swap in A.18.26(9.2/3) has no explicit mode on its parameters, but our style is to always give an explicit mode on procedure parameters. Should there be an explicit mode "in" here? (Yes.)

9) 6.2 is titled "Formal parameter modes". But nothing in this clause is about parameter modes other than the introduction 6.2(1) and the note 6.2(13). Should we do more? (Yes.)

Summary of Response

This AI corrects minor errors in the Standard.

1) Add type Frame to the examples in 3.10.

2) Add an example function with an "in out" parameter to 6.1.

3) Replace the second example of 5.2(20), the current one is not legal.

4) Delete excess "and" from 1.1(3/3).

5) "section" should be "clause" in 1.1.2(24/3).

6) "derived_type_declaration" should be "derived_type_definition" in the syntax font (and have a link) in 3.9(12.4/3).

7) Function Copy in A.18.25(10/3) should return type Tree, not type List.

8) Formal procedure Swap in A.18.26(9.2/3) should have an explicit mode "in" for its parameters.

9) Add two user notes in 6.2 to point at where the rules about parameter modes really are.

Corrigendum Wording

Replace 1.1(3):

The language provides rich support for real-time, concurrent programming, and includes facilities for multicore and multiprocessor programming. Errors can be signaled as exceptions and handled explicitly. The language also covers systems programming; this requires precise control over the representation of data and access to system-dependent properties. Finally, a predefined environment of standard packages is provided, including facilities for, among others, input-output, string manipulation, numeric elementary functions, and random number generation, and definition and use of containers.

by:

The language provides rich support for real-time, concurrent programming, and includes facilities for multicore and multiprocessor programming. Errors can be signaled as exceptions and handled explicitly. The language also covers systems programming; this requires precise control over the representation of data and access to system-dependent properties. Finally, a predefined environment of standard packages is provided, including facilities for, among others, input-output, string manipulation, numeric elementary functions, random number generation, and definition and use of containers.

Replace 1.1.2(24):

Each section is divided into subclauses that have a common structure. Each clause and subclause first introduces its subject. After the introductory text, text is labeled with the following headings:

by:

Each clause is divided into subclauses that have a common structure. Each clause and subclause first introduces its subject. After the introductory text, text is labeled with the following headings:

Replace 3.9(12.4):

The function Parent_Tag returns the tag of the parent type of the type whose tag is T. If the type does not have a parent type (that is, it was not declared by a derived_type_declaration), then No_Tag is returned.

by:

The function Parent_Tag returns the tag of the parent type of the type whose tag is T. If the type does not have a parent type (that is, it was not defined by a derived_type_definition), then No_Tag is returned.

Replace 3.10(22):

```
type Peripheral_Ref is not null access Peripheral;  -- see 3.8.1
type Binop_Ptr is access all Binary_Operation'Class;
                                                    -- general access-to-class-wide, see 3.9.1
```

by:

```
type Frame is access Matrix;     -- see 3.6
type Peripheral_Ref is not null access Peripheral;  -- see 3.8.1
type Binop_Ptr is access all Binary_Operation'Class;
                                                    -- general access-to-class-wide, see 3.9.1
```

Replace 5.2(20):

```
Writer := (Status => Open, Unit => Printer, Line_Count => 60);  -- see 3.8.1
Next_Car.all := (72074, null);      -- see 3.10.1
```

by:

```
Writer := (Status => Open, Unit => Printer, Line_Count => 60);  -- see 3.8.1
Next.all := (72074, null, Head);    -- see 3.10.1
```

Replace 6.1(39):

```
function Min_Cell(X : Link) return Cell;                  -- see 3.10.1
function Next_Frame(K : Positive) return Frame;           -- see 3.10
function Dot_Product(Left, Right : Vector) return Real;   -- see 3.6
```

by:

```
function Min_Cell(X : Link) return Cell;                  -- see 3.10.1
function Next_Frame(K : Positive) return Frame;           -- see 3.10
```

```
function Dot_Product(Left, Right : Vector) return Real;  -- see 3.6
function Find(B : aliased in out Barrel; Key : String) return Real;
                                                          -- see 4.1.5
```

Replace 6.2(13):

NOTES

6 A formal parameter of mode **in** is a constant view (see 3.3); it cannot be updated within the subprogram_body.

by:

NOTES

6 The mode of a formal parameter describes the direction of information transfer to or from the subprogram_body (see 6.1).

7 A formal parameter of mode **in** is a constant view (see 3.3); it cannot be updated within the subprogram_body.

8 A formal parameter of mode **out** might be uninitialized at the start of the subprogram_body (see 6.4.1).

Replace A.18.25(10):

```
function Copy (Source : Tree; Capacity : Count_Type := 0)
   return List;
```

by:

```
function Copy (Source : Tree; Capacity : Count_Type := 0)
   return Tree;
```

Replace A.18.26(9.2):

```
generic
   type Index_Type is (<>);
   with function Before (Left, Right : Index_Type) return Boolean;
   with procedure Swap (Left, Right : Index_Type);
procedure Ada.Containers.Generic_Sort
      (First, Last : Index_Type'Base);
pragma Pure(Ada.Containers.Generic_Sort);
```

by:

```
generic
   type Index_Type is (<>);
   with function Before (Left, Right : Index_Type) return Boolean;
   with procedure Swap (Left, Right : in Index_Type);
procedure Ada.Containers.Generic_Sort
      (First, Last : Index_Type'Base);
pragma Pure(Ada.Containers.Generic_Sort);
```

Discussion

1) The intent is that the examples in the Standard, taken as a whole, are complete: all of the types and subprograms are defined in (possibly other) examples. (This principle does *not* apply to examples in AARM notes.)

The use of type Frame in the declaration of Next_Frame violates this principle. Frame was declared in 3.8 in Ada 83, but it was removed from 3.10 (which is what 3.8 became) in Ada 95. This needs to be corrected.

Note that function Next_Frame is used in examples in 4.1.1 and 5.2, so removing it and replacing it by a more typical example is not a good idea.

2) The example functions all have "in" parameters, which shows only a subset of possibilities. We considered replacing Next_Frame (to fix the previous problem) to have a function with an "in out" parameter without making the example larger, but that would have required coming up with a new example for 4.1.1. So we just added a new function here.

The added example includes both an "in out" parameter and an "aliased" parameter, and is a rather typical use of these in a function that returns a generalized reference (similar to the uses in the containers packages).

[Editor's Note: We might want to add an abstract routine as well, or make this one abstract.]

3) The assignment requires a class-wide source object, as the target is class-wide. But it is not legal for an aggregate to have a class-wide type, so we have to qualify the aggregate with a specific type, and then convert that to a class-wide type.

That would require something like: Next_Car.all := Car'Class(Car'(72074, null))); -- see 3.10.1

This is ugly; moreover, these examples are intended to be very simple (without any conversions) and this surely is not.

Modifying the type of Next_Car would require care as other examples depend upon it. It wouldn't have to be class-wide to make those examples work, but then we'd need a new class-wide example. (And it's nice to show that you can still dereference and access components in a class-wide object.) So that doesn't seem to be a good idea.

Thus, we replace the example completely with one derived from types Cell and Link, and the object Next, conviniently defined directly above the Car example.

4) This is just a left-over word from when that sentence was rewritten at the last moment.

5) This use of "section" was incorrectly changed when "section" was changed into "clause" and "clause" into "subclause" at the direction of ITTF. The command inserted claimed to change it from "clause" to "section", exactly the reverse of what was supposed to happen. (The incorrect command was removed; the Ada 2012 AARM shows an incorrect change which does not match the Ada 2005 Standard.)

6) This is obviously missing formatting, which has been missing since the text was inserted into the Ada 2005 Consolidated Standard. But it probably was mising because there is no such thing as a "derived_type_declaration"; the syntax term is a "derived_type_definition". Then the word "declared" is also wrong, so it was changed to defined.

An alternative wording was considered: (that is, it is not a derived type) but this would be a bigger change, and it is not as clear that we're talking about a particular form of type definition. (Not everyone realizes that neither a private extension nor a formal derived type are a derived type, as they certainly look like one. See 3.4(1/2) for the definition of a derived type. This wording depends directly on that definition, as neither of the previously mentioned types has a parent type; the function ignores them when determining its result.)

7) This is pretty clearly a cut-and-paste error, as a similar bullet appears in each of the bounded container sections.

8) Our standard style has modes on all procedure parameters, so this one should as well.

9) This clause got its name from Ada 83, when it really did define the formal parameter modes. But the normative rules all have moved away. In order for the Standard to live up to its name ("Ada Reference Manual"), we at least should have some references to the actual normative rules. Since a note already provides one such reference, we add two more notes to give the other two for the other most important rules.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0119**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.2.4; 3.5.5; 3.8.1; 4.5.2; 4.6; 4.9.1; 5.4; 5.5; 13.9.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0119 Order of evaluation when multiple predicates apply

Working Reference Number AI12-0071-1

Question

3.2.4(6/3) says: The predicate of a subtype consists of all predicate specifications that apply, and-ed together; ...

and then we just talk about the evaluation of the predicate.

Should we define the order in which this evaluation is performed? (Yes.)

Summary of Response

If multiple predicates apply to a subtype when a check is required, then after checking the constraints and exclusions, the predicates are checked in an order that ensures that a predicate is not evaluated until any prior predicates evaluate to True. The order is unspecified for predicates that are defined or inherited as part of a single declaration.

The phrase "satisfies the predicates of a subtype" means that all relevant predicates evaluate to True.

Corrigendum Wording

Replace 3.2.4(4):

- For a (first) subtype defined by a derived type declaration, the predicates of the parent subtype and the progenitor subtypes apply.

by:

- For a (first) subtype defined by a type_declaration, the predicates of the parent subtype and the progenitor subtypes apply.

Delete 3.2.4(6):

The *predicate* of a subtype consists of all predicate specifications that apply, and-ed together; if no predicate specifications apply, the predicate is True (in particular, the predicate of a base subtype is True).

Insert before 3.2.4(30):

If predicate checks are enabled for a given subtype, then:

the new paragraphs:

If any of the above Legality Rules is violated in an instance of a generic unit, Program_Error is raised at the point of the violation.

To determine whether a value *satisfies the predicates* of a subtype *S*, the following tests are performed in the following order, until one of the tests fails, in which case the predicates are not satisfied and no further tests are performed, or all of the tests succeed, in which case the predicates are satisfied:

- the value is first tested to determine whether it satisfies any constraints or any null exclusion of *S*;
- then:
  - if *S* is a first subtype, the value is tested to determine whether it satisfies the predicates of the parent and progenitor subtypes (if any) of *S* (in an arbitrary order);
  - if *S* is defined by a subtype_indication, the value is tested to determine whether it satisfies the predicates of the subtype denoted by the subtype_mark of the subtype_indication;
- finally, if *S* is defined by a declaration to which one or more predicate specifications apply, the predicates are evaluated (in an arbitrary order) to test that all of them yield True for the given value.

Replace 3.2.4(31):

On every subtype conversion, the predicate of the target subtype is evaluated, and a check is performed that the predicate is True. This includes all parameter passing, except for certain parameters passed by reference, which are covered by the following rule: After normal completion and leaving of a subprogram, for each **in out** or **out** parameter that is passed by reference, the predicate of the subtype of the actual is evaluated, and a check is performed that the predicate is True. For an object created by an object_declaration with no explicit initialization expression, or by an uninitialized allocator, if any subcomponents have default_expressions, the predicate of the nominal subtype of the created object is evaluated, and a check is performed that the predicate is True. Assertions.Assertion_Error is raised if any of these checks fail.

by:

On every subtype conversion, a check is performed that the operand satisfies the predicates of the target subtype. This includes all parameter passing, except for certain parameters passed by reference, which are covered by the following rule: After normal completion and leaving of a subprogram, for each **in out** or **out** parameter that is passed by reference, a check is performed that the value of the parameter satisfies the predicates of the subtype of the actual. For an object created by an object_declaration with no explicit initialization expression, or by an uninitialized allocator, if any subcomponents have default_expressions, a check is performed that the value of the created object satisfies the predicates of the nominal subtype.

If any of the predicate checks fail, Assertion_Error is raised, unless the subtype whose directly-specified predicate aspect evaluated to False also has a directly-specified Predicate_Failure aspect. In that case, the specified Predicate_Failure expression is evaluated; if the evaluation of the Predicate_Failure expression propagates an exception occurrence, then this occurrence is propagated for the failure of the predicate check; otherwise, Assertion_Error is raised, with an associated message string defined by the value of the Predicate_Failure expression. In the absence of such a Predicate_Failure aspect, an implementation-defined message string is associated with the Assertion_Error exception.

Delete 3.2.4(32):

A value *satisfies* a predicate if the predicate is True for that value.

Delete 3.2.4(33):

If any of the above Legality Rules is violated in an instance of a generic unit, Program_Error is raised at the point of the violation.

Insert after 3.2.4(35):

6 A Static_Predicate, like a constraint, always remains True for all objects of the subtype, except in the case of uninitialized variables and other invalid values. A Dynamic_Predicate, on the other hand, is checked as specified above, but can become False at other times. For example, the predicate of a record subtype is not checked when a subcomponent is modified.

the new paragraph:

7 No predicates apply to the base subtype of a scalar type; every value of a scalar type *T* is considered to satisfy the predicates of *T*'Base.

Replace 3.5.5(7.1):

For every static discrete subtype S for which there exists at least one value belonging to S that satisfies any predicate of S, the following attributes are defined:

by:

For every static discrete subtype S for which there exists at least one value belonging to S that satisfies the predicates of S, the following attributes are defined:

Replace 3.5.5(7.2):

S'First_Valid
: S'First_Valid denotes the smallest value that belongs to S and satisfies the predicate of S. The value of this attribute is of the type of S.

by:

S'First_Valid
: S'First_Valid denotes the smallest value that belongs to S and satisfies the predicates of S. The value of this attribute is of the type of S.

Replace 3.5.5(7.3):

S'Last_Valid
: S'Last_Valid denotes the largest value that belongs to S and satisfies the predicate of S. The value of this attribute is of the type of S.

by:

S'Last_Valid
: S'Last_Valid denotes the largest value that belongs to S and satisfies the predicates of S. The value of this attribute is of the type of S.

Replace 3.8.1(10.1):

- A discrete_choice that is a subtype_indication covers all values (possibly none) that belong to the subtype and that satisfy the static predicate of the subtype (see 3.2.4).

by:

- A discrete_choice that is a subtype_indication covers all values (possibly none) that belong to the subtype and that satisfy the static predicates of the subtype (see 3.2.4).

Replace 3.8.1(15):

- If the discriminant is of a static constrained scalar subtype then, except within an instance of a generic unit, each non-**others** discrete_choice shall cover only values in that subtype that satisfy its predicate, and each value of that subtype that satisfies its predicate shall be covered by some discrete_choice (either explicitly or by **others**);

by:

- If the discriminant is of a static constrained scalar subtype then, except within an instance of a generic unit, each non-**others** discrete_choice shall cover only values in that subtype that satisfy its predicates, and each value of that subtype that satisfies its predicates shall be covered by some discrete_choice (either explicitly or by **others**);

Replace 4.5.2(29):

- The membership_choice is a subtype_mark, the tested type is scalar, the value of the simple_expression belongs to the range of the named subtype, and the predicate of the named subtype evaluates to True.

by:

- The membership_choice is a subtype_mark, the tested type is scalar, the value of the simple_expression belongs to the range of the named subtype, and the value satisfies the predicates of the named subtype.

Replace 4.5.2(30):

- The membership_choice is a subtype_mark, the tested type is not scalar, the value of the simple_expression satisfies any constraints of the named subtype, the predicate of the named subtype evaluates to True, and:

by:

- The membership_choice is a subtype_mark, the tested type is not scalar, the value of the simple_expression satisfies any constraints of the named subtype, the value satisfies the predicates of the named subtype, and:

Replace 4.6(51):

After conversion of the value to the target type, if the target subtype is constrained, a check is performed that the value satisfies this constraint. If the target subtype excludes null, then a check is made that the

value is not null. If predicate checks are enabled for the target subtype (see 3.2.4), a check is performed that the predicate of the target subtype is satisfied for the value.

by:

After conversion of the value to the target type, if the target subtype is constrained, a check is performed that the value satisfies this constraint. If the target subtype excludes null, then a check is made that the value is not null. If predicate checks are enabled for the target subtype (see 3.2.4), a check is performed that the value satisfies the predicates of the target subtype.

Replace 4.9.1(10):

- both subtypes are static, every value that satisfies the predicate of *S1* also satisfies the predicate of *S2*, and it is not the case that both types each have at least one applicable predicate specification, predicate checks are enabled (see 11.4.2) for *S2*, and predicate checks are not enabled for *S1*.

by:

- both subtypes are static, every value that satisfies the predicates of *S1* also satisfies the predicates of *S2*, and it is not the case that both types each have at least one applicable predicate specification, predicate checks are enabled (see 11.4.2) for *S2*, and predicate checks are not enabled for *S1*.

Replace 5.4(7):

- If the *selecting_*expression is a name (including a type_conversion, qualified_expression, or function_call) having a static and constrained nominal subtype, then each non-**others** discrete_choice shall cover only values in that subtype that satisfy its predicate (see 3.2.4), and each value of that subtype that satisfies its predicate shall be covered by some discrete_choice (either explicitly or by **others**).

by:

- If the *selecting_*expression is a name (including a type_conversion, qualified_expression, or function_call) having a static and constrained nominal subtype, then each non-**others** discrete_choice shall cover only values in that subtype that satisfy its predicates (see 3.2.4), and each value of that subtype that satisfies its predicates shall be covered by some discrete_choice (either explicitly or by **others**).

Replace 5.5(9):

For the execution of a loop_statement with the iteration_scheme being **for** loop_parameter_specification, the loop_parameter_specification is first elaborated. This elaboration creates the loop parameter and elaborates the discrete_subtype_definition. If the discrete_subtype_definition defines a subtype with a null range, the execution of the loop_statement is complete. Otherwise, the sequence_of_statements is executed once for each value of the discrete subtype defined by the discrete_subtype_definition that satisfies the predicate of the subtype (or until the loop is left as a consequence of a transfer of control). Prior to each such iteration, the corresponding value of the discrete subtype is assigned to the loop parameter. These values are assigned in increasing order unless the reserved word **reverse** is present, in which case the values are assigned in decreasing order.

by:

For the execution of a loop_statement with the iteration_scheme being **for** loop_parameter_specification, the loop_parameter_specification is first elaborated. This elaboration creates the loop parameter and elaborates the discrete_subtype_definition. If the discrete_subtype_definition defines a subtype with a null range, the execution of the loop_statement is complete. Otherwise, the sequence_of_statements is executed once for each value of the discrete subtype defined by the discrete_subtype_definition that satisfies the predicates of the subtype (or until the loop is left as a consequence of a transfer of control). Prior to each such iteration, the corresponding value of the discrete subtype is assigned to the loop parameter. These values are assigned in increasing order unless the reserved word **reverse** is present, in which case the values are assigned in decreasing order.

Replace 13.9.2(3):

X'Valid

Yields True if and only if the object denoted by X is normal, has a valid representation, and the predicate of the nominal subtype of X evaluates to True. The value of this attribute is of the predefined type Boolean.

by:

X'Valid

Yields True if and only if the object denoted by X is normal, has a valid representation, and then, if the preceding conditions hold, the value of X also satisfies the predicates of the nominal subtype of X. The value of this attribute is of the predefined type Boolean.

Replace 13.9.2(12):

23 X'Valid is not considered to be a read of X; hence, it is not an error to check the validity of invalid data.

by:

23 Determining whether X is normal and has a valid representation as part of the evaluation of X'Valid is not considered to include an evaluation of X; hence, it is not an error to check the validity of an object that is invalid or abnormal. Determining whether X satisfies the predicates of its nominal subtype may include an evaluation of X, but only after it has been determined that X has a valid representation.

If X is volatile, the evaluation of X'Valid is considered a read of X.

Response

(See !summary.)

Discussion

The obvious solution is to explicitly state that the order is unspecified. That's how other assertions work. However, there are several other factors occurring here that don't occur for other assertions.

Most important is the realization that predicates are created by refinement (all predicates are evaluated, each possibly raising their own exception as proposed in AI12-0054-2). Preconditions, by contrast are defined by replacement -- Pre can only be replaced, not modified. (Admittedly, class-wide preconditions are different in this sense -- they might in fact have a similar problem.)

In particular, the expectation is that the constraints/exclusions/predicates of a "parent" subtype hold in the child subtype. If we have:

```
subtype Small is Integer range 1..100;
subtype Tiny is Small range 1..10;
```

We expect Tiny to be a subrange of Small (and we'd get Constraint_Error if it isn't). We want a similar dynamic to hold for predicates. After all, if we didn't want the predicate to hold, we just would make the second subtype a subtype of the base type directly.

This is critical if the exception raised by the failure of a particular predicate is specified (as in AI12-0054-2). If the parts of the declarations are evaluated in a totally arbitrary order, the "wrong" exception could be raised. AI12-0054-2 recommends using multiple subtypes to have different exceptions raised for different failures (see the examples below). If the order is unspecified, then the exact exception raised would not be well-defined, which would make it harder to portably convert existing interfaces into contract assertions.

Moreover, in extreme cases, it could cause an exception to be raised when no exception should be raised at all. Consider:

```
subtype Nonnull_Item_Ptr is not null Item_Ptr;
subtype Nonempty_Item_Ptr is Nonnull_Item_Ptr
   with Dynamic_Predicate => Nonempty_Item_Ptr.Count > 0;

if Ptr in Nonempty_Item_Ptr then -- (1)
```

The membership at (1) should not raise an exception. However, if the predicate was evaluated before the null exclusion, and Ptr was null, the evaluation of the predicate as part of the membership would raise Constraint_Error.

Similar effects are possible using just predicates, or using constraints and predicates. Another example is (using the routines defined in Text_IO):

```
subtype Open_File_Type is File_Type
   with Dynamic_Predicate => Is_Open (Open_File_Type),
        Predicate_Failure => raise Status_Error;
subtype Read_File_Type is Open_File_Type
   with Dynamic_Predicate => Mode (Read_File_Type) = In_File;
        Predicate_Failure => raise Mode_Error with "Can't read file: " & Name
(Read_File_Type);

if My_File in Read_File_Type then -- (2)
```

If the predicate of Read_File_Type is evaluated first, then (2) would raise Status_Error if My_File is not open, whereas the expected result is for the membership to return False.

Thus we define that the constraints and exclusions are evaluated first when checking whether a value satisfies a subtype, the predicates are then checked in an order that ensures that a predicate is not evaluated until any prior predicates evaluate to True. This has no effect on the order of checks when multiple checks are needed (as in a subprogram call with multiple parameters); that is still unspecified.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0120**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.2.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0120 Wording problems with predicates

Working Reference Number AI12-0099-1

Question

(1) Consider the case of a task type with progenitors: task type Tsk is new Intf with ... end Tsk;

Since Tsk does not have a predicate specification, 3.2.4(8/3-11/3) do not apply. 3.2.4(12/3) doesn't apply either, as Tsk is not a derived type declaration. 3.2.4(13/3) does not apply as Tsk is not a subtype_indication. So I can only conclude that 3.2.4(14/3) applies.

That of course means that any predicates of Intf will never be checked as part of Tsk, other than in memberships, no matter what the assertion policy is (anywhere).

It's pretty clear that 3.2.4(12/3) is intended to apply to this case. As with 3.2.4(4/3) [now 4/4], the original author seems to have missed that a task type or protected type is not a derived type declaration, but that they can have progenitors and need to be treated similarly to a derived type declaration.

Do we need to change "derived type declaration" to "type declaration" in 3.2.4(12/3)? (Yes.)

(2) There does not seem to be a definition of the term "logical operator" (no underscore) as used in RM 3.2.4(20/3). In particular, it does not appear to include the "not" operator. (An English term that is not otherwise defined is usually taken to mean the same as the associated syntax term, and "logical_operator" does not include "not". The index entry for logical operator has "See also not operator" but that seems hardly a normative definition!

Thus it appears that "not" cannot be used in a "predicate-static" expression. Should we fix the wording to make it clear that "not" is allowed in predicate-static expressions? (Yes.)

Summary of Response

(1) 3.2.4(12/3) applies to a type declared by any type of declaration, not just a derived type declaration. In particular, it applies to task types and protected types.

(2) "not" is an allowed operator in a predicate-static expression.

Corrigendum Wording

Replace 3.2.4(4):

- For a (first) subtype defined by a type_declaration, the predicates of the parent subtype and the progenitor subtypes apply.

by:

- For a (first) subtype defined by a type declaration, any predicates of parent or progenitor subtypes apply.

Replace 3.2.4(12):

- If a subtype is defined by a derived type declaration that does not include a predicate specification, then predicate checks are enabled for the subtype if and only if predicate checks are enabled for at least one of the parent subtype and the progenitor subtypes;

by:

- If a subtype is defined by a type declaration that does not include a predicate specification, then predicate checks are enabled for the subtype if and only if any predicate checks are enabled for parent or progenitor subtypes;

Replace 3.2.4(20):

- a call to a predefined boolean logical operator, where each operand is predicate-static;

by:

- a call to a predefined boolean operator **and**, **or**, **xor**, or **not**, where each operand is predicate-static;

Discussion

(1) AI12-0071-1 made a similar change to 3.2.4(4/4), to correct essentially the same problem in a different rule. We clearly missed the fact that there are two such rules.

Note that "type declaration" is not the syntax term "type_declaration" in either this rule or 3.2.4(4/4), as we need to include the anonymous task type associated with a single_task_declaration (which is not a "type_declaration"), and similarly for single_protected_declaration. We change 3.2.4(4/4) this way because the editor failed to understand the meaning of the Steve Baird comment in AI12-0071-1 and thus got it wrong.

(2) As noted in the question, the typical policy for the Standard is to have otherwise undefined terms to have the same meaning as the syntax term that results from swapping spaces for underscores. In this case, that means that "not" is not a logical operation.

It certainly is illogical for "not" to be excluded from logical operators, but that's been true since at least Ada 80, so changing that after more than 33 years is more likely to introduce bugs into the Standard than to fix any. So we opt to just fix the predicate-static wording to avoid any confusion.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0121**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.2.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0121 Aspect Predicate_Failure

Working Reference Number AI12-0054-2

Question

AI12-0022-1, "Raise expressions for specifying the exception raised for an assertion" added the ability to control which exception is raised by failure of various assertions. This is an important capability, because it allows one to change interfaces to use assertions while preserving compatibility.

However, this feature doesn't work properly for predicates, because predicates are evaluated by membership tests, so we would get spurious failures in such membership tests if the predicate expression included a raise expression. Should these spurious failures be removed from the language, so that there is a mechanism for specifying the exception raised by a predicate failure? (Yes.)

Summary of Response

The aspect Predicate_Failure defines what happens when a predicate check fails. It does NOT apply when evaluating memberships and the Valid attribute.

Corrigendum Wording

Insert after 3.2.4(14):

- Otherwise, predicate checks are disabled for the given subtype.

the new paragraphs:

For a subtype with a directly-specified predicate aspect, the following additional language-defined aspect may be specified with an aspect_specification (see 13.1.1):

Predicate_Failure
This aspect shall be specified by an expression, which determines the action to be performed when a predicate check fails because a directly-specified predicate aspect of the subtype evaluates to False, as explained below.

Name Resolution Rules

The expected type for the Predicate_Failure expression is String.

Replace 3.2.4(31):

On every subtype conversion, the predicate of the target subtype is evaluated, and a check is performed that the predicate is True. This includes all parameter passing, except for certain parameters passed by reference, which are covered by the following rule: After normal completion and leaving of a subprogram, for each **in out** or **out** parameter that is passed by reference, the predicate of the subtype of the actual is evaluated, and a check is performed that the predicate is True. For an object created by an object_declaration with no explicit initialization expression, or by an uninitialized allocator, if any subcomponents have default_expressions, the predicate of the nominal subtype of the created object is evaluated, and a check is performed that the predicate is True. Assertions.Assertion_Error is raised if any of these checks fail.

by:

On every subtype conversion, the predicate of the target subtype is evaluated, and a check is performed that the predicate is True. This includes all parameter passing, except for certain parameters passed by reference, which are covered by the following rule: After normal completion and leaving of a subprogram, for each **in out** or **out** parameter that is passed by reference, the predicate of the subtype of the actual is evaluated, and a check is performed that the predicate is True. For an object created by an object_declaration with no explicit initialization expression, or by an uninitialized allocator, if any subcomponents have default_expressions, the predicate of the nominal subtype of the created object is evaluated, and a check is performed that the predicate is True.

If any of the predicate checks fail, Assertion_Error is raised, unless the subtype whose directly-specified predicate aspect evaluated to False also has a directly-specified Predicate_Failure aspect. In that case, the specified Predicate_Failure expression is evaluated; if the evaluation of the Predicate_Failure expression propagates an exception occurrence, then this occurrence is propagated for the failure of the predicate check; otherwise, Assertion_Error is raised, with an associated message string defined by the value of the Predicate_Failure expression. In the absence of such a Predicate_Failure aspect, an implementation-defined message string is associated with the Assertion_Error exception.

Insert after 3.2.4(35):

6 A Static_Predicate, like a constraint, always remains True for all objects of the subtype, except in the case of uninitialized variables and other invalid values. A Dynamic_Predicate, on the other hand, is checked as specified above, but can become False at other times. For example, the predicate of a record subtype is not checked when a subcomponent is modified.

the new paragraphs:

7 Predicate_Failure expressions are never evaluated during the evaluation of a membership test (see 4.5.2) or Valid attribute (see 13.9.2).

8 A Predicate_Failure expression can be a raise_expression (see 11.3).

Examples

```
subtype Basic_Letter is Character -- See A.3.2 for "basic letter".
   with Static_Predicate => Basic_Letter in 'A'..'Z' | 'a'..'z' | 'Æ' | 'æ'
| 'Ð' | 'ð' | 'Þ' | 'þ' | 'ß';

subtype Even_Integer is Integer
   with Dynamic_Predicate => Even_Integer mod 2 = 0,
        Predicate_Failure => "Even_Integer must be a multiple of 2";
```

Text_IO (see A.10.1) could have used predicates to describe some common exceptional conditions as follows:

```
with Ada.IO_Exceptions;
package Ada.Text_IO is

   type File_Type is limited private;

   subtype Open_File_Type is File_Type
      with Dynamic_Predicate => Is_Open (Open_File_Type),
           Predicate_Failure => raise Status_Error with "File not open";
   subtype Input_File_Type is Open_File_Type
      with Dynamic_Predicate => Mode (Input_File_Type) = In_File,
           Predicate_Failure => raise Mode_Error with "Cannot read file: "
&
              Name (Input_File_Type);
   subtype Output_File_Type is Open_File_Type
      with Dynamic_Predicate => Mode (Output_File_Type) /= In_File,
           Predicate_Failure => raise Mode_Error with "Cannot write file: "
&
              Name (Output_File_Type);

   ...

   function Mode (File : in Open_File_Type) return File_Mode;
   function Name (File : in Open_File_Type) return String;
   function Form (File : in Open_File_Type) return String;

   ...

   procedure Get (File : in Input_File_Type; Item : out Character);

   procedure Put (File : in Output_File_Type; Item : in Character);

   ...

   -- Similarly for all of the other input and output subprograms.
```

Response

(See !summary.)

Discussion

Note that Predicate_Failure is not involved when a predicate is evaluated in a membership or Valid attribute. This is how we get our cake and eat it too in this case.

Originally the Predicate_Failure was suggested as a way to be able to specify a different exception to be raised when failing a predicate check, without causing an exception to be raised when the predicate expression were evaluated as part of a membership test or Valid attribute reference. But then it was realized that the Predicate_Failure expression could be something other than simply a raise expression. In particular, it was suggested that it could specify the string to be associated with the predicate failure, even when still raising Assertion_Error. The first example in the wording above is a simple example, using Predicate_Failure to specify the message string to be associated with the Assertion_Error.

The expression of the Predicate_Failure aspect can be as simple as a string literal if all that is desired is to change the message; note that the current instance of the subtype is visible in the Predicate_Failure expression -- this can be useful in the exception message. On the other hand, the expression could be a complicated function call to log the bad value, for example.

If the evaluation of the expression raises an exception (an especially if it is a raise expression), the exception will propagate normally, thus achieving the effect of the predicate raising an expression different from Assertion_Error.

If it is necessary to raise multiple exceptions for different failures, or have distinct messages depending on which predicate fails, it is necessary to define multiple subtypes. For an example, see the Text_IO example in the wording above.

One could imagine defining similar aspects for the other kinds of contract assertions such as Pre and Post. However, there is no counterpart to the membership test or Valid attribute that causes the trouble with predicates. But defining such aspects would not be hard.

A simpler alternative would be to simply have a Predicate_Failure_Exception aspect, which just specifies the exception. In that case, however, we've lost the ability to specify an exception message. Thus we prefer the given solution.

NOTE: The example in AI12-0022-1 should not use a predicate expression as an example of where raise_expression should be used. That part of AI12-0022-1 should be replaced by the following.

Example: Imagine the following routine in a GUI library:

```
procedure Show_Window (Window : in out Root_Window);
   -- Shows the window.
   -- Raises Not_Valid_Error if Window is not valid.
```

We would like to be able to use a predicate to check the comment. With the "raise_expression" (and the Predicate_Failure aspect, see AI12-0054-2), we can do this without changing the semantics:

```
subtype Valid_Root_Window is Root_Window
   with Dynamic_Predicate => Is_Valid (Valid_Root_Window),
        Predicate_Failure => raise Not_Valid_Error;

procedure Show_Window (Window : in out Valid_Root_Window);
   -- Shows the window.
```

If we didn't include the Predicate_Failure aspect with a raise_expression here, using the predicate would change the exception raised on this failure. That could cause the exception to fall into a different handler than currently, which is unlikely to be acceptable.

We could have used a precondition on Show_Window instead of defining a predicate. In that case, we'd use the raise_expression directly in the precondition to raise the correct exception:

```
procedure Show_Window (Window : in out Valid_Root_Window)
   with Pre => Is_Valid (Window) or else raise Not_Valid_Error;
   -- Shows the window.
```

or perhaps a conditional expression is preferable:

```
procedure Show_Window (Window : in out Valid_Root_Window)
   with Pre => (if not Is_Valid (Window) then raise Not_Valid_Error);
   -- Shows the window.
```

Similarly, the various Containers packages in Ada could use predicates or preconditions in this way to make some of the needed checks; but that can only be done if the semantics remains unchanged (raising Program_Error and Constraint_Error, not Assertion_Error). (The !proposal also shows how this could be used in Text_IO and other I/O packages.)

End replacement for AI12-0022-1.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0122**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.2.4; 4.4; 4.5.2; 4.9; 8.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0122 Ambiguity in syntax for membership expression removed

Working Reference Number AI12-0039-1

Question

It appears that the Ada 2012 syntax for membership expressions introduces ambiguities.

Assume A, B, C, and D are all objects of type Boolean.

Consider A in B and C This could be interpreted as equivalent to either of these valid Ada expressions: (A in B) and C --or as A in (B and C)

Similarly:

A in B in C | D can be interpreted as A in (B in C) | D --or as A in (B in C | D)

And:

A in B | C and D can be interpreted as (A in B | C) and D --or as A in B | (C and D)

Should the syntax be corrected to eliminate these ambiguities? (Yes.)

Summary of Response

The choices of a membership are syntactically simple_expressions, not choice_expressions.

Corrigendum Wording

Replace 3.2.4(17):

- a membership test whose simple_expression is the current instance, and whose membership_choice_list meets the requirements for a static membership test (see 4.9);

by:

- a membership test whose *tested*_simple_expression is the current instance, and whose membership_choice_list meets the requirements for a static membership test (see 4.9);

Replace 4.4(3):

relation ::=
  simple_expression [relational_operator simple_expression]
| simple_expression [**not**] **in** membership_choice_list

by:

relation ::=
  simple_expression [relational_operator simple_expression]
| *tested*_simple_expression [**not**] **in** membership_choice_list

Replace 4.4(3.2):

membership_choice ::= choice_expression | range | subtype_mark

by:

membership_choice ::= *choice*_simple_expression | range | subtype_mark

Replace 4.5.2(3.1):

If the tested type is tagged, then the simple_expression shall resolve to be of a type that is convertible (see 4.6) to the tested type; if untagged, the expected type for the simple_expression is the tested type. The expected type of a choice_expression in a membership_choice, and of a simple_expression of a range in a membership_choice, is the tested type of the membership operation.

by:

If the tested type is tagged, then the *tested*_simple_expression shall resolve to be of a type that is convertible (see 4.6) to the tested type; if untagged, the expected type for the *tested*_simple_expression is the tested type. The expected type of a *choice*_simple_expression in a membership_choice, and of a simple_expression of a range in a membership_choice, is the tested type of the membership operation.

Replace 4.5.2(4):

For a membership test, if the simple_expression is of a tagged class-wide type, then the tested type shall be (visibly) tagged.

by:

For a membership test, if the *tested*_simple_expression is of a tagged class-wide type, then the tested type shall be (visibly) tagged.

Replace 4.5.2(4.1):

If a membership test includes one or more choice_expressions and the tested type of the membership test is limited, then the tested type of the membership test shall have a visible primitive equality operator.

by:

If a membership test includes one or more *choice*_simple_expressions and the tested type of the membership test is limited, then the tested type of the membership test shall have a visible primitive equality operator.

Replace 4.5.2(27):

For the evaluation of a membership test using **in** whose membership_choice_list has a single membership_choice, the simple_expression and the membership_choice are evaluated in an arbitrary order; the result is the result of the individual membership test for the membership_choice.

by:

For the evaluation of a membership test using **in** whose membership_choice_list has a single membership_choice, the *tested*_simple_expression and the membership_choice are evaluated in an arbitrary order; the result is the result of the individual membership test for the membership_choice.

Replace 4.5.2(27.1):

For the evaluation of a membership test using **in** whose membership_choice_list has more than one membership_choice, the simple_expression of the membership test is evaluated first and the result of the operation is equivalent to that of a sequence consisting of an individual membership test on each membership_choice combined with the short-circuit control form **or else**.

by:

For the evaluation of a membership test using **in** whose membership_choice_list has more than one membership_choice, the *tested*_simple_expression of the membership test is evaluated first and the result of the operation is equivalent to that of a sequence consisting of an individual membership test on each membership_choice combined with the short-circuit control form **or else**.

Replace 4.5.2(28.1):

- The membership_choice is a choice_expression, and the simple_expression is equal to the value of the membership_choice. If the tested type is a record type or a limited type, the test uses the primitive equality for the type; otherwise, the test uses predefined equality.

by:

- The membership_choice is a *choice*_simple_expression, and the *tested*_simple_expression is equal to the value of the membership_choice. If the tested type is a record type or a limited type, the test uses the primitive equality for the type; otherwise, the test uses predefined equality.

Replace 4.5.2(28.2):

- The membership_choice is a range and the value of the simple_expression belongs to the given range.

by:

- The membership_choice is a range and the value of the *tested*_simple_expression belongs to the given range.

Replace 4.5.2(29):

- The membership_choice is a subtype_mark, the tested type is scalar, the value of the simple_expression belongs to the range of the named subtype, and the predicate of the named subtype evaluates to True.

by:

- The membership_choice is a subtype_mark, the tested type is scalar, the value of the *tested*_simple_expression belongs to the range of the named subtype, and the predicate of the named subtype evaluates to True.

Replace 4.5.2(30):

- The membership_choice is a subtype_mark, the tested type is not scalar, the value of the simple_expression satisfies any constraints of the named subtype, the predicate of the named subtype evaluates to True, and:

by:

- The membership_choice is a subtype_mark, the tested type is not scalar, the value of the *tested*_simple_expression satisfies any constraints of the named subtype, the predicate of the named subtype evaluates to True, and:

Replace 4.5.2(30.1):

if the type of the simple_expression is class-wide, the value has a tag that identifies a type covered by the tested type;

by:

if the type of the *tested*_simple_expression is class-wide, the value has a tag that identifies a type covered by the tested type;

Replace 4.5.2(30.2):

- if the tested type is an access type and the named subtype excludes null, the value of the simple_expression is not null;

by:

- if the tested type is an access type and the named subtype excludes null, the value of the *tested*_simple_expression is not null;

Replace 4.5.2(30.3):

- if the tested type is a general access-to-object type, the type of the simple_expression is convertible to the tested type and its accessibility level is no deeper than that of the tested type; further, if the designated type of the tested type is tagged and the simple_expression is nonnull, the tag of the object designated by the value of the simple_expression is covered by the designated type of the tested type.

by:

- if the tested type is a general access-to-object type, the type of the *tested*_simple_expression is convertible to the tested type and its accessibility level is no deeper than that of the tested type; further, if the designated type of the tested type is tagged and the *tested*_simple_expression is nonnull, the tag of the object designated by the value of the *tested*_simple_expression is covered by the designated type of the tested type.

Replace 4.9(11):

- a membership test whose simple_expression is a static expression, and whose membership_choice_list consists only of membership_choices that are either static choice_expressions, static ranges, or subtype_marks that denote a static (scalar or string) subtype;

by:

- a membership test whose *tested_*simple_expression is a static expression, and whose membership_choice_list consists only of membership_choices that are either static *choice_*simple_expressions, static ranges, or subtype_marks that denote a static (scalar or string) subtype;

Replace 4.9(32.6):

- a choice_expression (or a simple_expression of a range that occurs as a membership_choice of a membership_choice_list) of a static membership test that is preceded in the enclosing membership_choice_list by another item whose individual membership test (see 4.5.2) statically yields True.

by:

- a *choice_*simple_expression (or a simple_expression of a range that occurs as a membership_choice of a membership_choice_list) of a static membership test that is preceded in the enclosing membership_choice_list by another item whose individual membership test (see 4.5.2) statically yields True.

Replace 8.6(27.1):

Other than for the simple_expression of a membership test, if the expected type for a name or expression is not the same as the actual type of the name or expression, the actual type shall be convertible to the expected type (see 4.6); further, if the expected type is a named access-to-object type with designated type *D1* and the actual type is an anonymous access-to-object type with designated type *D2*, then *D1* shall cover *D2*, and the name or expression shall denote a view with an accessibility level for which the statically deeper relationship applies; in particular it shall not denote an access parameter nor a stand-alone access object.

by:

Other than for the *tested_*simple_expression of a membership test, if the expected type for a name or expression is not the same as the actual type of the name or expression, the actual type shall be convertible to the expected type (see 4.6); further, if the expected type is a named access-to-object type with designated type *D1* and the actual type is an anonymous access-to-object type with designated type *D2*, then *D1* shall cover *D2*, and the name or expression shall denote a view with an accessibility level for which the statically deeper relationship applies; in particular it shall not denote an access parameter nor a stand-alone access object.

Discussion

The basic intent was that the choice of a membership was syntactically equivalent to the other uses of a choice (such as in a case statement). Choice_expression was introduced to minimize the incompatibility with existing choices in case statements. Unfortunately, we failed to notice that doing so made the grammar ambiguous.

Clearly, the constituents of a membership choice either have to have higher precedence than a membership or have to be enclosed in parentheses. If a membership or logical operator is used in a membership choice, it has to be parenthesized.

This is most easily accomplished by making a membership choice a simple expression; this gives the proper precedence.

It is annoying that choices for memberships and case statements are subtly different. Had this been a concern in 1994, probably choices would never have been changed from simple_expression (in Ada 83) to expression (in Ada 95) to choice_expression (in Ada 2012). But today, minimizing compatibility issues with Ada 95 has to be more important than having everything exactly the same.

Note that this change has no effect the interpretation of on any syntax or construct that existed in Ada 2005 or before; it only could change the interpretation of new Ada 2012 expressions (of which there are hopefully very few taking advantage of this ambiguity).

Unfortunately, changing the syntax this way makes all of the rules that talk about "the simple_expression" and "a choice_expression" in 4.5.2 ambiguous. As such, we have to give these syntax items prefixes to eliminate any confusion. Thus we talk about "tested_simple_expression" and "choice_simple_expression".

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0123**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **3.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0123 Add Object'Image

Working Reference Number AI12-0124-1

Question

GNAT has the attribute Obj'Img, which is very useful for debugging and testing as it does not require looking up the name of the subtype of Obj. This is commonly used in GNAT programs. Should standard Ada should have something similar? (Yes.)

Summary of Response

Allow the prefix of the Image attribute to be an object.

Corrigendum Wording

Insert after 3.5(55):

> For the evaluation of a call on S'Value for an enumeration subtype S, if the sequence of characters of the parameter (ignoring leading and trailing spaces) has the syntax of an enumeration literal and if it corresponds to a literal of the type of S (or corresponds to the result of S'Image for a value of the type), the result is the corresponding enumeration value; otherwise, Constraint_Error is raised. For a numeric subtype S, the evaluation of a call on S'Value with *Arg* of type String is equivalent to a call on S'Wide_Wide_Value for a corresponding *Arg* of type Wide_Wide_String.

the new paragraphs:

For a prefix X that denotes an object of a scalar type (after any implicit dereference), the following attributes are defined:

X'Wide_Wide_Image
: X'Wide_Wide_Image denotes the result of calling function S'Wide_Wide_Image with *Arg* being X, where S is the nominal subtype of X.

X'Wide_Image
: X'Wide_Image denotes the result of calling function S'Wide_Image with *Arg* being X, where S is the nominal subtype of X.

X'Image
: X'Image denotes the result of calling function S'Image with *Arg* being X, where S is the nominal subtype of X.

Discussion

We use an overloading of the existing attribute Image rather than following GNAT's lead of Img. Our understanding is that GNAT used Img rather than Image simply because 4.1.4(12/1) requires that implementation-defined attributes have different names than language-defined ones. Moreover, Ada generally does not use abbreviations, and this abbreviation would only save two characters.

This addition is compatible, as subtypes do not allow overloading. Thus, if an object X hides a subtype X, X'Image is currently illegal (and this would be a straight extension). If a subtype X hides an object X, no change would occur.

This should be implementable relatively easily, as the Size attribute already allows both subtypes and objects as the prefix, so there is a similar attribute to use as a model.

Note that the prefix of an attribute has to be a name, but any expression can be wrapped in a qualification or type conversion in order to make it a name. (Of course, in that case, the advantage of not having to know the subtype name has disappeared.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0124**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.5.9; 11.3; 11.4.1; J.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0124 Eliminate ambiguities in raise expression and derived type syntax

Working Reference Number AI12-0152-1

Question

There appear to be a number of ambiguities in the Ada syntax, mostly involving raise expressions.

(A) Consider the expression:

raise Program_Error with A and B

This could be interpreted as

(raise Program_Error with A) and B or raise Program_Error with (A and B)

The Ada expression grammar does not appear to make a choice in this case.

(B) Consider the object declaration:

Nasty : Natural := raise TBD_Error with Atomic;

This could be interpreted as:

Nasty : Natural := (raise TBD_Error) with Atomic;

or

Nasty : Natural := (raise TBD_Error with Atomic);

A similar problem occurs in component_declarations.

(C) Consider:

Val : String := "Oops";

A := (raise TBD_Error with Val);

This is a classic raise_expression. Unfortunately, it's also an extension_aggregate, made clearer with parens:

A := ((raise TBD_Error) with Val);

It should be possible to distinguish these syntactically.

(D) consider the following insane type declaration:

Atomic : String := "Gotcha!";

```
type Fun is new My_Decimal_Type digits raise TBD_Error with Atomic;
```

This is using a digits_constraint (I purposely used the non-obsolescent one) in a subtype_indication in a derived type declaration.

This can be interpreted as:

```
type Fun is new My_Decimal_Type digits (raise TBD_Error with Atomic);
```

or

```
type Fun is new My_Decimal_Type digits (raise TBD_Error) with Atomic;
```

(the latter being an aspect specification for aspect Atomic, lest you've forgotten).

Ada 2005 introduced a similar problem:

```
   A, B : constant Some_Modular_Type := ...;

type Nutso is new Some_Type digits A and B with private;
```

This could be interpreted as:

```
type Nutso is new Some_Type digits (A and B) with private;
```

or the "and B" could be interpreted as an interface list.

This of course can't be legal (at least not until we have tagged real types), but it does confuse a parser. And it is not far from the legal declaration:

```
type Nutso2 is new Some_Type digits A and B with Volatile;
```

which we surely do have to parse with the current grammar.

We also have a similar problem with type declarations using digits and delta constraints:

```
type Bad1 is digits raise TBD_Error with Atomic;
type Bad2 is delta  raise TBD_Error with Atomic;
type Bad3 is digits 5 delta raise TBD_Error with Atomic;

Should these things be fixed? (Yes.)
```

Summary of Response

Modify the Ada grammar to eliminate ambiguities.

Corrigendum Wording

Replace 3.5.9(5):

digits_constraint ::=
digits *static*_expression [range_constraint]

by:

digits_constraint ::=
digits *static*_simple_expression [range_constraint]

Replace 3.5.9(18):

For a digits_constraint on a decimal fixed point subtype with a given *delta*, if it does not have a range_constraint, then it specifies an implicit range –(10**_D_–1)*_delta_ .. +(10**_D_–1)*_delta_, where *D* is the value of the expression. A digits_constraint is *compatible* with a decimal fixed point subtype if the value of the expression is no greater than the *digits* of the subtype, and if it specifies (explicitly or implicitly) a range that is compatible with the subtype.

by:

For a digits_constraint on a decimal fixed point subtype with a given *delta*, if it does not have a range_constraint, then it specifies an implicit range –(10**_D_–1)*_delta_ .. +(10**_D_–1)*_delta_, where *D* is the value of the simple_expression. A digits_constraint is *compatible* with a decimal fixed point subtype if the value of the simple_expression is no greater than the *digits* of the subtype, and if it specifies (explicitly or implicitly) a range that is compatible with the subtype.

Replace 3.5.9(19):

The elaboration of a digits_constraint consists of the elaboration of the range_constraint, if any. If a range_constraint is given, a check is made that the bounds of the range are both in the range –(10**_D_–

1)*delta* .. +(10**D*–1)**delta*, where *D* is the value of the (static) expression given after the reserved word **digits**. If this check fails, Constraint_Error is raised.

by:

The elaboration of a digits_constraint consists of the elaboration of the range_constraint, if any. If a range_constraint is given, a check is made that the bounds of the range are both in the range –(10**D*–1)**delta* .. +(10**D*–1)**delta*, where *D* is the value of the (static) simple_expression given after the reserved word **digits**. If this check fails, Constraint_Error is raised.

Insert after 11.3(2):

raise_statement ::= **raise**; |
raise *exception_*name [**with** *string_*expression];

the new paragraphs:

raise_expression ::= **raise** *exception_*name [**with** *string_*simple_expression]

If a raise_expression appears within the expression of one of the following contexts, the raise_expression shall appear within a pair of parentheses within the expression:

- object_declaration;
- modular_type_definition;
- floating_point_definition;
- ordinary_fixed_point_definition;
- decimal_fixed_point_definition;
- default_expression;
- ancestor_part.

Replace 11.3(3.1):

The expression, if any, in a raise_statement, is expected to be of type String.

by:

The *string_*expression or *string_*simple_expression, if any, of a raise_statement or raise_expression is expected to be of type String.

Replace 11.3(4):

To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception_*name, the named exception is raised. Similarly, for the evaluation of a raise_expression, the named exception is raised. In both of these cases, if a *string_*expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

by:

To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception_*name, the named exception is raised. Similarly, for the evaluation of a raise_expression, the named exception is raised. In both of these cases, if a *string_*expression or *string_*simple_expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

Replace 11.4.1(10.1):

Exception_Message returns the message associated with the given Exception_Occurrence. For an occurrence raised by a call to Raise_Exception, the message is the Message parameter passed to Raise_Exception. For the occurrence raised by a raise_statement with an *exception_*name and a *string_*expression, the message is the *string_*expression. For the occurrence raised by a raise_statement with an *exception_*name but without a *string_*expression, the message is a string giving implementation-defined information about the exception occurrence. For an occurrence originally

raised in some other manner (including by the failure of a language-defined check), the message is an unspecified string. In all cases, Exception_Message returns a string with lower bound 1.

by:

Exception_Message returns the message associated with the given Exception_Occurrence. For an occurrence raised by a call to Raise_Exception, the message is the Message parameter passed to Raise_Exception. For the occurrence raised by a raise_statement or raise_expression with an *exception_*name and a *string_*expression or *string_*simple_expression, the message is the *string_*expression or *string_*simple_expression. For the occurrence raised by a raise_statement or raise_expression with an *exception_*name but without a *string_*expression or *string_*simple_expression, the message is a string giving implementation-defined information about the exception occurrence. For an occurrence originally raised in some other manner (including by the failure of a language-defined check), the message is an unspecified string. In all cases, Exception_Message returns a string with lower bound 1.

Replace J.3(2):

delta_constraint ::=
delta *static_*expression [range_constraint]

by:

delta_constraint ::=
delta *static_*simple_expression [range_constraint]

Replace J.3(3):

The expression of a delta_constraint is expected to be of any real type.

by:

The simple_expression of a delta_constraint is expected to be of any real type.

Replace J.3(4):

The expression of a delta_constraint shall be static.

by:

The simple_expression of a delta_constraint shall be static.

Replace J.3(7):

A subtype_indication with a subtype_mark that denotes an ordinary fixed point subtype and a delta_constraint defines an ordinary fixed point subtype with a *delta* given by the value of the expression of the delta_constraint. If the delta_constraint includes a range_constraint, then the ordinary fixed point subtype is constrained by the range_constraint.

by:

A subtype_indication with a subtype_mark that denotes an ordinary fixed point subtype and a delta_constraint defines an ordinary fixed point subtype with a *delta* given by the value of the simple_expression of the delta_constraint. If the delta_constraint includes a range_constraint, then the ordinary fixed point subtype is constrained by the range_constraint.

Replace J.3(8):

A subtype_indication with a subtype_mark that denotes a floating point subtype and a digits_constraint defines a floating point subtype with a requested decimal precision (as reflected by its Digits attribute) given by the value of the expression of the digits_constraint. If the digits_constraint includes a range_constraint, then the floating point subtype is constrained by the range_constraint.

by:

A subtype_indication with a subtype_mark that denotes a floating point subtype and a digits_constraint defines a floating point subtype with a requested decimal precision (as reflected by its Digits attribute) given by the value of the simple_expression of the digits_constraint. If the digits_constraint includes a range_constraint, then the floating point subtype is constrained by the range_constraint.

Replace J.3(9):

A delta_constraint is *compatible* with an ordinary fixed point subtype if the value of the expression is no less than the *delta* of the subtype, and the range_constraint, if any, is compatible with the subtype.

by:

A delta_constraint is *compatible* with an ordinary fixed point subtype if the value of the simple_expression is no less than the *delta* of the subtype, and the range_constraint, if any, is compatible with the subtype.

Replace J.3(10):

A digits_constraint is *compatible* with a floating point subtype if the value of the expression is no greater than the requested decimal precision of the subtype, and the range_constraint, if any, is compatible with the subtype.

by:

A digits_constraint is *compatible* with a floating point subtype if the value of the simple_expression is no greater than the requested decimal precision of the subtype, and the range_constraint, if any, is compatible with the subtype.

Discussion

Almost all of these are "dangling else" problems. They didn't occur in the original proposal for raise_expression (which proposed that it work like a conditional expression). However, early reviewers thought that required too many parens. Thus we decided during a meeting to drop the parens altogether, without considering the effect on the syntax of other declarations. A more judicious approach to dropping parens would have been better, even if it would not have appeased everyone. (Consistency with other statements turned into expressions would have been a powerful reason for keeping the parens.) However, as this feature has been in use for several years, we do not want to make as drastic a change as that would be.

For (A), we chose to make the optional string into a simple_expression. Since the expression has to be of type string, this has a minimal difference, as someone would have to declare a logical or relational operator with a return type of type String for there to be any possibility of noticing the change. In that unlikely case, parens would be needed around the string expression.

For (B), we note that we cannot tolerate any incompatibility with any existing expressions, as initialized object declarations are very common. Even unusual expressions probably have occurred somewhere. As such, we've adopted a change that only requires parenthesizing raise expressions in such a context, as that can only affect code that used raise_expressions before they were formally defined. We also only require extra parentheses in cases where the raise_expression would have no parentheses at all; if it is inside of any parenthesized expression, aggregate, parameter list, or the like, no additional parentheses are required.

For (C), a raise expression cannot be legally given as the ancestor expression of an extension aggregate, unless qualified, as it does not determine a unique specific tagged type (raise_expressions match any type). Thus, we only need to make an extension aggregate somehow different than a raise_expression. We adopt the same solution as for (B), requiring the raise expression to appear in parentheses in an extension aggregate. In this case, that means that

```
A := (raise TBD_Error with Val);
```

is always a raise_expression, no matter what Val is, and

```
A := (raise TBD_Error with Comp => Val);
```

is syntactically illegal (either parens are needed around "raise TBD_Error", or "Comp =>" is extra).

For (D), we can note that any use of a raise expression in a fixed or float definition, or in a digits or delta constraint, is illegal, as a raise expression is not static. Thus the same solution as in (B) is sufficient for the raise_expression problem.

However, the Ada 2005-introduced ambiguity with "and" requires a larger change to digits and delta constraints.

For both digit_constraint and delta_constraint, changing *static*_expression to *static*_simple_expression eliminates the problem.

For delta_constraint, this only requires putting extra parentheses around expressions that are necessarily illegal, so this is completely compatible. In particular, user-defined operators are not allowed as they are not static, memberships, predefined relational operators, and short circuit operations always return Boolean (which cannot be "any real type"), and predefined logical operators only could be of a Boolean, modular, or array type (none of which would match "any real type").

For digits_constraint, all of the same is true, with one exception: logical operators of modular types would be allowed. So there is a very unlikely incompatibility with this change. For there to be a problem, all of the above would need to be true: (1) A modular type is declared in the program, with at least one static constant; (possible) (2) A digits_constraint would have to be used as a subtype_indication (unlikely, most such uses are obsolescent, and the others are for the rarely used decimal fixed types); (3) The digits value would have to be created from an expression involving "and", "or", or "xor" (unlikely, most digits values are literals or named numbers). (4) The digits expression is not parenthesized. (probable) In that doubly unlikely case, the digits expression would have to be parenthesized.

The case in question would look like:

```
type Modular is mod 2**8;
Num : constant Modular := 7;

type Dec is digits 7 delta 0.01;

subtype Really is Dec digits Num and 3; -- Legal in Ada 95 & 2005, but not
                                        -- allowed by syntax change.
```

The subtype would have to be written:

```
subtype Really is Dec digits (Num and 3); -- OK.
```

Alternatives considered:

We considered using a grammar change rather than an English rule for the majority of these cases. (This mentioned in the AARM notes for the chosen solution.) That would look like:

Add in 4.4:

initial_expression ::= initial_relation {and initial_relation} | initial_relation {or initial_relation} | initial_relation {xor initial_relation} | initial_relation {and then initial_relation} | initial_relation {or else initial_relation}

initial_relation ::= simple_expression [relational_operator simple_expression] | tested_simple_expression simple_expression [not] in membership_choice_list

Replace 3.3.1(2/3) by:

> object_declaration ::= defining_identifier_list : [aliased] [constant] subtype_indication [:= initial_expression] [aspect_specification]; | defining_identifier_list : [aliased] [constant] access_definition [:= initial_expression] [aspect_specification]; | defining_identifier_list : [aliased] [constant] array_type_definition [:= initial_expression] [aspect_specification]; | single_task_declaration | single_protected_declaration

Replace 3.5.4(4) by:

> modular_type_definition ::= mod *static*_initial_expression

Replace 3.5.7(2) by:

> floating_point_definition ::= digits static_initial_expression [real_range_specification]

Replace 3.5.9(3-4) by:

ordinary_fixed_point_definition ::= delta static_initial_expression real_range_specification

decimal_fixed_point_definition ::= delta static_initial_expression digits static_initial_expression [real_range_specification]

Replace 3.7(6) by:

default_expression ::= initial_expression

Replace 4.3.2(3) by:

ancestor_part ::= initial_expression | subtype_mark

However, a large number of Semantic and Legality Rules that refer to these syntactic "expression"s would also need to be changed. (Just as was necessary in the cases where we did change the grammar.) That seemed like too much change.

It would be appealing to make the English-language syntax rule apply to all raise expressions. This would be more consistent with the way conditional expressions and quantified expressions are handled - it makes sense for all "expression statements" to be treated similarly syntactically. The effect would be as if "initial_expression" above was used in all contexts where an expression appears that are not themselves part of a larger expression. For example, the expanded rule would apply in if conditions and in return expressions.

We did not do this as raise_expression as originally defined has been implemented and in use in at least one compiler for several years. While stand-alone raise expressions are unlikely in most contexts, some uses likely exist and there doesn't seem to be any reason to break those. In particular, the construct "return raise TBD_Error;" (or some other convenient exception) is suggested for use when providing a body-to-be-defined later for a function. This neatly gets around the requirement for "at least one return statement" in a function body. It's also likely that some preconditions or postconditions are defined using "or else" followed by a raise expression. Adopting the English-language rule globally would require both of those to be surrounded by parentheses.

--

Looking in the other direction, we considered a number of rules that would change the rules of AI12-0022-1 less.

One suggestion was that parentheses only be required around a raise_expression when the optional with part was included. This would be a bit less change, but it also would cause a maintenance issue if the "with expr" was added after the initial raise_expression was compiled. In that case, the programmer would get an annoying "parens required" after adding the with part, while the initial compile was legal.

Similarly, a suggestion that the extra parens be required only when the expression precedes some other "with" (an aspect specification or the rest of an extension aggregate) seems to have maintenance issues. Again, adding an aspect specification during maintenance would trigger an annoying "parens required". In both of these cases, it's unlikely that the programmer would remember the parentheses requirement, and such a rule would increase Ada's reputation for obscure annoyances.

The wildest of these suggestions was to determine whether a "with" was part of a preceding raise_expression or part of an aspect specification by determining if the first identifier was a possible aspect_mark. The objection to that is that the list of possible aspect_marks is implementation-defined, thus the meaning of a legal Ada program could differ between implementations.

For instance, imagine that compiler A has a Boolean aspect Exact_Size_Only, and compiler B does not. Then

Exact_Size_Only : constant String := "Exact size required!";

```
Obj : Boolean := Func or else raise Some_Error with Exact_Size_Only;
```

would mean

```
Obj : Boolean := Func or else (raise Some_Error) with Exact_Size_Only;
```

when compiled with compiler A, and

```
Obj : Boolean := Func or else (raise Some_Error with Exact_Size_Only);
```

when compiled with compiler B.

While unlikely, this is intolerable; we want implementation-defined stuff to possibly change the program from legal to illegal (or vice-versa), not between two very different meanings.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0125**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.5.9; 4.1.4; 6.1.1; 6.3.1; 6.4.1; 7.3.2; 11.3; A; A.18.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0125 Corrections from the Corrigendum Editorial Review

Working Reference Number AI12-0159-1

Question

(1) There does not appear to be any resolution rule that applies to a digits_constraint. Should there be one? (Yes.)

(2) In 6.1.1(38/4) and 7.3.2(22.1/4), are the commas in the right place? (No.)

(3) There is a missing "the" in 7.3.2(21/4). Should it be added? (Yes.)

(4) In A.18.10(2/4), "consists of" would be better than "comprises of". The comma after root node seems misplaced. Should these be changed? (Yes.)

(5) In A(3.1/4), we have both "input-output" and "input/output". One of them must be wrong, right? (Yes.)

(6) In 11.3(4.1/4), "rather than" sounds odd. Wouldn't "instead of" be better?

(7) In 11.3(3/4), the original "if any" text makes more sense. Should it be replaced? (Yes.)

(8) The type of a raise_expression is too vague. We don't want it to match anything in any circumstance, else it becomes impossible to enforce Legality Rules that depend on the type of an expression. (For instance, in an extension aggregate ((raise Something) with ...), does (raise Something) have a specific tagged type?) Should this be fixed? (Yes.)

(9) The last two sentences of 6.1.1(7/4) are non-normative discussion that better belongs in the AARM. Should it be moved there? (Yes.)

(10) In 6.3.1(12/4), the first and third (existing) cases start "for a", but the (new) second case starts just "a". Should a third "for" be added? (Yes.) Similarly 6.3.1(13/4) should include "for", right? (Yes.)

(11) 6.4.1(6.26-6.29/4) look like a continuation of the rules above them 6.4.1(6.18-6.25/3), but they're not. Would these be better placed after 6.4.1(5)? (Yes.)

(12) In 6.4.1(13.1-4/4), the "except in the following case:" is awkward and ambiguous. Can we do better? (Yes.)

(13) 6.4.1(26.2/4) says "is of an anonymous access, ...". Is the word "type" missing here? (Yes.)

(14) 6.1.1(37/4) starts out with For any subprogram or entry call S (including dispatching calls),

which suggests that S is the call (i.e., an expression or statement), not the callee (i.e., a subprogram). The rest of the paragraph goes on to treat S as though it is the callee.

Contrast this with the correct phrasing in paragraph 38: The class-wide precondition check for a call to a subprogram or entry S...

Should this be fixed? (Yes.)

(15) 4.1.4(9/4) mentions "without constraint or null exclusion" without mentioning predicates, which presumably are also ignored here. Should they be mentioned? (Yes.)

Summary of Response

Various issues with normative wording found during editorial reviews are addressed.

Corrigendum Wording

Insert after 3.5.9(6):

For a type defined by a fixed_point_definition, the *delta* of the type is specified by the value of the expression given after the reserved word **delta**; this expression is expected to be of any real type. For a type defined by a decimal_fixed_point_definition (a *decimal* fixed point type), the number of significant decimal digits for its first subtype (the *digits* of the first subtype) is specified by the expression given after the reserved word **digits**; this expression is expected to be of any integer type.

the new paragraph:

The simple_expression of a digits_constraint is expected to be of any integer type.

Replace 4.1.4(9):

An attribute_reference denotes a value, an object, a subprogram, or some other kind of program entity. For an attribute_reference that denotes a value or an object, if its type is scalar, then its nominal subtype is the base subtype of the type; if its type is tagged, its nominal subtype is the first subtype of the type; otherwise, its nominal subtype is a subtype of the type without any constraint or null_exclusion. Similarly, unless explicitly specified otherwise, for an attribute_reference that denotes a function, when its result type is scalar, its result subtype is the base subtype of the type, when its result type is tagged, the result subtype is the first subtype of the type, and when the result type is some other type, the result subtype is a subtype of the type without any constraint or null_exclusion.

by:

An attribute_reference denotes a value, an object, a subprogram, or some other kind of program entity. Unless explicitly specified otherwise, for an attribute_reference that denotes a value or an object, if its type is scalar, then its nominal subtype is the base subtype of the type; if its type is tagged, its nominal subtype is the first subtype of the type; otherwise, its nominal subtype is a subtype of the type without any constraint, null_exclusion, or predicate. Similarly, unless explicitly specified otherwise, for an attribute_reference that denotes a function, when its result type is scalar, its result subtype is the base subtype of the type, when its result type is tagged, the result subtype is the first subtype of the type, and when the result type is some other type, the result subtype is a subtype of the type without any constraint, null_exclusion, or predicate.

Replace 6.1.1(7):

Within the expression for a Pre'Class or Post'Class aspect for a primitive subprogram of a tagged type *T*, a name that denotes a formal parameter of type *T* is interpreted as having type *T*'Class. Similarly, a name that denotes a formal access parameter of type access-to-*T* is interpreted as having type access-to-*T*'Class. This ensures that the expression is well-defined for a primitive subprogram of a type descended from *T*.

by:

Within the expression for a Pre'Class or Post'Class aspect for a primitive subprogram *S* of a tagged type *T*, a name that denotes a formal parameter (or *S*'Result) of type *T* is interpreted as though it had a (notional) type *NT* that is a formal derived type whose ancestor type is *T*, with directly visible primitive operations. Similarly, a name that denotes a formal access parameter (or *S*'Result) of type access-to-*T* is interpreted as having type access-to-*NT*. The result of this interpretation is that the only operations that can be applied to such names are those defined for such a formal derived type.

Replace 6.1.1(26):

X'Old

For each X'Old in a postcondition expression that is enabled, a constant is implicitly declared at the beginning of the subprogram or entry. The constant is of the type of X and is initialized to the result of evaluating X (as an expression) at the point of the constant declaration. The value of X'Old in the postcondition expression is the value of this constant; the type of X'Old is the type of X. These implicit constant declarations occur in an arbitrary order.

by:

X'Old

Each X'Old in a postcondition expression that is enabled denotes a constant that is implicitly declared at the beginning of the subprogram body, entry body, or accept statement.

The implicitly declared entity denoted by each occurrence of X'Old is declared as follows:

- If X is of an anonymous access type defined by an access_definition *A* then

  ```
  X'Old : constant A := X;
  ```

- If X is of a specific tagged type *T* then

  ```
  anonymous : constant T'Class := T'Class (X);

  X'Old : T renames T(anonymous);
  ```

  where the name X'Old denotes the object renaming.

- Otherwise

  ```
  X'Old : constant S := X;
  ```

  where *S* is the nominal subtype of X. This includes the case where the type of *S* is an anonymous array type or a universal type.

The nominal subtype of X'Old is as implied by the above definitions. The expected type of the prefix of an Old attribute is that of the attribute. Similarly, if an Old attribute shall resolve to be of some type, then the prefix of the attribute shall resolve to be of that type.

Replace 6.1.1(37):

For any subprogram or entry call (including dispatching calls), the checks that are performed to verify specific precondition expressions and specific and class-wide postcondition expressions are determined by those for the subprogram or entry actually invoked. Note that the class-wide postcondition expressions verified by the postcondition check that is part of a call on a primitive subprogram of type *T* includes all class-wide postcondition expressions originating in any progenitor of *T*, even if the primitive subprogram called is inherited from a type *T1* and some of the postcondition expressions do not apply to the corresponding primitive subprogram of *T1*.

by:

For any call to a subprogram or entry *S* (including dispatching calls), the checks that are performed to verify specific precondition expressions and specific and class-wide postcondition expressions are determined by those for the subprogram or entry actually invoked. Note that the class-wide postcondition expressions verified by the postcondition check that is part of a call on a primitive subprogram of type *T* includes all class-wide postcondition expressions originating in any progenitor of *T*, even if the primitive subprogram called is inherited from a type *T1* and some of the postcondition expressions do not apply to the corresponding primitive subprogram of *T1*. Any operations within a class-wide postcondition expression that were resolved as primitive operations of the (notional) formal derived type *NT*, are in the evaluation of the postcondition bound to the corresponding operations of the type identified by the controlling tag of the call on *S*. This applies to both dispatching and non-dispatching calls on *S*.

Replace 6.1.1(38):

The class-wide precondition check for a call to a subprogram or entry consists solely of checking the class-wide precondition expressions that apply to the denoted callable entity (not necessarily the one that is invoked).

by:

The class-wide precondition check for a call to a subprogram or entry *S* consists solely of checking the class-wide precondition expressions that apply to the denoted callable entity (not necessarily to the one that is invoked). Any operations within such an expression that were resolved as primitive operations of the (notional) formal derived type *NT* are in the evaluation of the precondition bound to the corresponding operations of the type identified by the controlling tag of the call on *S*. This applies to both dispatching and non-dispatching calls on *S*.

Replace 6.3.1(12):

- The default calling convention is *protected* for a protected subprogram, and for an access-to-subprogram type with the reserved word **protected** in its definition.

by:

- The default calling convention is *protected* for a protected subprogram, for a prefixed view of a subprogram with a synchronization kind of By_Protected_Procedure, and for an access-to-subprogram type with the reserved word **protected** in its definition.

Replace 6.3.1(13):

- The default calling convention is *entry* for an entry.

by:

- The default calling convention is *entry* for an entry and for a prefixed view of a subprogram with a synchronization kind of By_Entry.

Replace 6.4.1(13.1):

- For a scalar type that has the Default_Value aspect specified, the formal parameter is initialized from the value of the actual, without checking that the value satisfies any constraint or any predicate;

by:

- For a scalar type that has the Default_Value aspect specified, the formal parameter is initialized from the value of the actual, without checking that the value satisfies any constraint or any predicate. Furthermore, if the actual parameter is a view conversion and either
  - there exists no type (other than a root numeric type) that is an ancestor of both the target type and the type of the operand of the conversion; or
  - the Default_Value aspect is unspecified for the type of the operand of the conversion

  then Program_Error is raised;

Replace 7.3.2(5):

Within an invariant expression, the identifier of the first subtype of the associated type denotes the current instance of the type. Within an invariant expression associated with type *T*, the type of the current instance is *T* for the Type_Invariant aspect and *T*'Class for the Type_Invariant'Class aspect.

by:

Within an invariant expression, the identifier of the first subtype of the associated type denotes the current instance of the type. Within an invariant expression for the Type_Invariant aspect of a type *T*, the type of this current instance is *T*. Within an invariant expression for the Type_Invariant'Class aspect of a type *T*, the type of this current instance is interpreted as though it had a (notional) type *NT* that is a visible formal derived type whose ancestor type is *T*. The effect of this interpretation is that the only operations that can be applied to this current instance are those defined for such a formal derived type.

Replace 7.3.2(21):

If performing checks is required by the Invariant or Invariant'Class assertion policies (see 11.4.2) in effect at the point of corresponding aspect specification applicable to a given type, then the respective invariant expression is considered *enabled*.

by:

If performing checks is required by the Type_Invariant or Type_Invariant'Class assertion policies (see 11.4.2) in effect at the point of the corresponding aspect specification applicable to a given type, then the respective invariant expression is considered *enabled*.

Insert after 7.3.2(22):

The invariant check consists of the evaluation of each enabled invariant expression that applies to *T*, on each of the objects specified above. If any of these evaluate to False, Assertions.Assertion_Error is raised at the point of the object initialization, conversion, or call. If a given call requires more than one evaluation of an invariant expression, either for multiple objects of a single type or for multiple types with invariants, the evaluations are performed in an arbitrary order, and if one of them evaluates to False, it is not specified whether the others are evaluated. Any invariant check is performed prior to copying back any by-copy **in out** or **out** parameters. Invariant checks, any postcondition check, and any constraint or predicate checks associated with **in out** or **out** parameters are performed in an arbitrary order.

the new paragraph:

For an invariant check on a value of type *T1* based on a class-wide invariant expression inherited from an ancestor type *T*, any operations within the invariant expression that were resolved as primitive operations

of the (notional) formal derived type *NT* are bound to the corresponding operations of type *T1* in the evaluation of the invariant expression for the check on *T1*.

Replace 11.3(3):

The name, if any, in a raise_statement shall denote an exception. A raise_statement with no *exception_*name (that is, a *re-raise statement*) shall be within a handler, but not within a body enclosed by that handler.

by:

The *exception_*name, if any, of a raise_statement or raise_expression shall denote an exception. A raise_statement with no *exception_*name (that is, a *re-raise statement*) shall be within a handler, but not within a body enclosed by that handler.

Replace 11.3(3.1):

The expression, if any, in a raise_statement, is expected to be of type String.

by:

A *string_*expression of a raise_statement or raise_expression is expected to be of type String.

The expected type for a raise_expression shall be any single type.

Replace 11.3(4):

To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception_*name, the named exception is raised. If a *string_*expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

by:

To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception_*name, the named exception is raised. Similarly, for the evaluation of a raise_expression, the named exception is raised. In both of these cases, if a *string_*expression or *string_*simple_expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

NOTES

1 If the evaluation of a *string_*expression or *string_*simple_expression raises an exception, that exception is propagated instead of the one denoted by the *exception_*name of the raise_statement or raise_expression.

Replace A(3):

The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on the same subprogram perform as specified, so long as all parameters that could be passed by reference denote nonoverlapping objects.

by:

The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on any language-defined subprogram perform as specified, so long as all objects that are denoted by parameters that could be passed by reference or designated by parameters of an access type are nonoverlapping.

For the purpose of determining whether concurrent calls on text input-output subprograms are required to perform as specified above, when calling a subprogram within Text_IO or its children that implicitly operates on one of the default input-output files, the subprogram is considered to have a parameter of Current_Input or Current_Output (as appropriate).

Replace A.18.10(2):

A multiway tree container object manages a tree of internal *nodes*, each of which contains an element and pointers to the parent, first child, last child, next (successor) sibling, and previous (predecessor) sibling internal nodes. A cursor designates a particular node within a tree (and by extension the element

contained in that node, if any). A cursor keeps designating the same node (and element) as long as the node is part of the container, even if the node is moved within the container.

by:

A multiway tree container object manages a tree of *nodes*, consisting of a *root node* and a set of *internal nodes*; each internal node contains an element and pointers to the parent, first child, last child, next (successor) sibling, and previous (predecessor) sibling internal nodes. A cursor designates a particular node within a tree (and by extension the element contained in that node, if any). A cursor keeps designating the same node (and element) as long as the node is part of the container, even if the node is moved within the container.

Discussion

(1) This appears to be an oversight.

(2) Comma madness. :-)

(3) Helps to edit. ;-) This is in part of the paragraph introduced by a last-minute change in the Ada 2012 wording.

(4) "comprising of" is just weird. The comma after "root node" is also weird, but it is very important to the meaning (as the contents of the "root node" are defined in the next paragraph, it's not the list following "internal node"). Thus we break the sentence into two parts so that we no longer need the comma, by repeating the part about the internal nodes.

(5) The standard usually uses "input-output" to describe files, libraries, etc., not "input/output".

(6) The RM/AARM uses "rather than" more often than "instead of". (175 occurrences to 55) Still, "instead of" sounds better to the Editor's ear as well as the commenter. The commenter suggests that "rather than" best applies to a substitute action, and "instead of" best applies to a substitute object. For example: He kicked rather than hit the ball. He hit the ball instead of the balloon. And this sentence is talking about replacing the object (an exception) rather than replacing the action (propagation).

(7) The wording was changed from the original "if any" wording (which is now the new wording), to the "An" wording, during discussion during the Stockholm ARG meeting. The objection appears to have been that the exception_name is not optional for a raise_expression, so "if any" is weird. However, the "An" wording seems to imply that there can be more than one, so it isn't clearly better.

(8) The suggested wording is similar to the way aggregates and string literals work, other entities that themselves do not provide a type.

This is incompatible in weird cases. For instance, this bans raise expressions from being the operand of a type conversion (a qualified expression should be used instead). This appears to include the case where a type conversion is distributed to the operands of a conditional expression: T(if A then B else raise Constraint_Error) is illegal by this rule.

(9) The last sentence of 6.1.1(7/4) comes unmodified from Ada 2012 text.

We considered adding a user note, but of course that is not near the rules in question. Thus it would come too late to really be helpful in explaining the changes.

(10) These are unintended inconsistencies in wording. Most of the 6.3.1 bullets are written in the form of "the calling convention for <something> is <some-convention>".

(11) These rules do get lost after the anti-aliasing rules, they should have more visibility.

The best place to move these rules would be after 6.4.1(6.4/3), since that is the last rule other than the anti-aliasing rules. But inserting them there would change all of the paragraph numbers following (we don't allow inserts of inserts for paragraph numbers). We don't want to do that.

So, after 6.4.1(5) is the only sensible alternative for the placement of these rules.

(12) Since this additional text specifies an additional, non-optional check, it doesn't really matter to which it applies, other than the part about "for a scalar type with Default_Value specified". Whether a parameter that fails this check is initialized cannot be determined (the parameter could not be read), and initialization cannot raise an exception. Thus separating it as suggested (or even making it a separate paragraph under the same bullet) is OK.

(13) "anonymous access" by itself doesn't mean anything, it needs to modify something. In this case, that's a type. Arguably, that can be inferred from the fact that an access_definition (which declares an anonymous type) is involved, but it's better to be explicit.

(14) The questioner is correct about the change in meaning of S. The wording should be fixed.

(15) We certainly don't mean to include predicates in this case, so we need to explicit mention that they don't apply (two places). This is clearly an omission.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0126**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **3.9.3; 7.3.2; 13.11.6; A.18.2; A.18.32; B.1; N**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0126 More presentation errors in Ada 2012

Working Reference Number AI12-0080-1

Question

1) The following appears to be legal:

```
package Pack1 is
    type T1 is abstract tagged record ... end record;
    function Create (N : Integer) return T1 is abstract;
end Pack1;

package Pack2 is
    type T2 is new Pak1.T1 with private;
private
    type T2 is new Pak1.T1 with null record;
end Pack2;
```

as Create is "a function with a controlling result" inherited by a private extension, as described in 3.9.3(6). But Create is abstract, and we don't want abstract subprograms of nonabstract tagged types. Should this be fixed? (Yes.)

2) B.1(50) says that the examples are of "interfacing pragmas", but all of the following examples are of "interfacing aspects". Should this be fixed? (Yes.)

3) N(21.2/3) start "A invariant". That should be "An invariant", right? (Yes.)

4) The Edge renaming in A.18.32(29/3) and A.18.32(31/3) ends with ".all". But a generalized reference is itself a dereference; it's not a pointer that needs to be dereferenced. Should the ".all" be deleted? (Yes.)

5) 7.3.2(21/3) talks about the "Invariant or Invariant'Class policy". But of course the policy is for Type_Invariant. Should this say "Type_Invariant"? (Yes.)

6) A.18.2(168/2) describes a Prepend with 3 parameters, including Count. But the equivalence in A.18.2(169/2) does not reference Count, and the package specification at A.18.2(44/2) does not have a Count parameter. Is this parameter a mistake? (Yes.)

7) The alignment code in 13.11.6(28/3) assumes that Pool.Storage is properly aligned at the maximum alignment. Moreover, it does not check for nonsense alignments. Should there be some indication that this code isn't realistic? (Yes.)

Summary of Response

This AI corrects minor errors in the Standard.

1) An abstract function inherited for a null extension still requires overriding.

2) The examples in B.1 are of interfacing aspects, of course.

3) "An invariant" rather than "A invariant".

4) Drop ".all" from the Edge renaming in the containers example.

5) "Invariant" should be "Type_Invariant" in 7.3.2 as there is no such thing as an "Invariant policy".

6) Drop the Count parameter from Prepend in A.18.2(168/2).

7) Add a check for the maximum supported alignment in 13.11.6(28/3).

Corrigendum Wording

Replace 3.9.3(6):

- Otherwise, the subprogram shall be overridden with a nonabstract subprogram or, in the case of a private extension inheriting a function with a controlling result, have a full type that is a null extension; for a type declared in the visible part of a package, the overriding may be either in the visible or the private part. Such a subprogram is said to *require overriding*. However, if the type is a generic formal type, the subprogram need not be overridden for the formal type itself; a nonabstract version will necessarily be provided by the actual type.

by:

- Otherwise, the subprogram shall be overridden with a nonabstract subprogram or, in the case of a private extension inheriting a nonabstract function with a controlling result, have a full type that is a null extension; for a type declared in the visible part of a package, the overriding may be either in the visible or the private part. Such a subprogram is said to *require overriding*. However, if the type is a generic formal type, the subprogram need not be overridden for the formal type itself; a nonabstract version will necessarily be provided by the actual type.

Replace 7.3.2(21):

If performing checks is required by the Invariant or Invariant'Class assertion policies (see 11.4.2) in effect at the point of corresponding aspect specification applicable to a given type, then the respective invariant expression is considered *enabled*.

by:

If performing checks is required by the Type_Invariant or Type_Invariant'Class assertion policies (see 11.4.2) in effect at the point of corresponding aspect specification applicable to a given type, then the respective invariant expression is considered *enabled*.

Replace 13.11.6(28):

```
      -- Correct the alignment if necessary:
      Pool.Next_Allocation := Pool.Next_Allocation +
         ((-Pool.Next_Allocation) mod Alignment);
      if Pool.Next_Allocation + Size_In_Storage_Elements >
         Pool.Pool_Size then
         raise Storage_Error; -- Out of space.
      end if;
      Storage_Address := Pool.Storage (Pool.Next_Allocation)'Address;
      Pool.Next_Allocation :=
         Pool.Next_Allocation + Size_In_Storage_Elements;
   end Allocate_From_Subpool;
```

by:

```
      -- Check for the maximum supported alignment, which is the alignment of the storage area:
      if Alignment > Pool.Storage'Alignment then
         raise Program_Error;
      end if;
      -- Correct the alignment if necessary:
      Pool.Next_Allocation := Pool.Next_Allocation +
         ((-Pool.Next_Allocation) mod Alignment);
      if Pool.Next_Allocation + Size_In_Storage_Elements >
         Pool.Pool_Size then
         raise Storage_Error; -- Out of space.
      end if;
      Storage_Address := Pool.Storage (Pool.Next_Allocation)'Address;
      Pool.Next_Allocation :=
         Pool.Next_Allocation + Size_In_Storage_Elements;
   end Allocate_From_Subpool;
```

Replace A.18.2(168):

```
   procedure Prepend (Container : in out Vector;
                      New_Item  : in     Vector;
                      Count     : in     Count_Type := 1);
```

by:

```
procedure Prepend (Container : in out Vector;
                   New_Item  : in     Vector);
```

Replace A.18.32(29):

```
for C in G (Next).Iterate loop
   declare
      E : Edge renames G (Next)(C).all;
   begin
      if not Reached(E.To) then
         ...
      end if;
   end;
end loop;
```

by:

```
for C in G (Next).Iterate loop
   declare
      E : Edge renames G (Next)(C);
   begin
      if not Reached(E.To) then
         ...
      end if;
   end;
end loop;
```

Replace A.18.32(31):

```
declare
   L : Adjacency_Lists.List renames G (Next);
   C : Adjacency_Lists.Cursor := L.First;
begin
   while Has_Element (C) loop
      declare
         E : Edge renames L(C).all;
      begin
         if not Reached(E.To) then
            ...
         end if;
      end;
      C := L.Next (C);
   end loop;
end;
```

by:

```
declare
   L : Adjacency_Lists.List renames G (Next);
   C : Adjacency_Lists.Cursor := L.First;
begin
   while Has_Element (C) loop
      declare
         E : Edge renames L(C);
      begin
         if not Reached(E.To) then
            ...
         end if;
      end;
      C := L.Next (C);
   end loop;
end;
```

Replace B.1(50):

Example of interfacing pragmas:

by:

Example of interfacing aspects:

Replace N(21.2):

> **Invariant.** A invariant is an assertion that is expected to be True for all objects of a given private type when viewed from outside the defining package.

by:

> **Invariant.** An invariant is an assertion that is expected to be True for all objects of a given private type when viewed from outside the defining package.

Insert after N(41):

> **Type.** Each object has a type. A *type* has an associated set of values, and a set of *primitive operations* which implement the fundamental aspects of its semantics. Types are grouped into *categories*. Most language-defined categories of types are also *classes* of types.

the new paragraph:

> **Type Invariant.** See Invariant.

Discussion

1) This problem was noted in the e-mail appendix to AI95-00391-1 (which originally defined this wording), but for some reason the fix was never applied. It obviously would make no sense to allow an abstract routine to be inherited but not be overridden for a nonabstract type (of any kind).

2) Since the clause title is "Interfacing aspects", it's pretty clear what the intent is!

3) "An" should proceed nouns that start with vowel sounds, as in this case.

4) Since a generalized reference is not (usually) a pointer, it can't (usually) be dereferenced. So ".all" doesn't make sense in this example.

5) Since there is no such thing as an "Invariant" policy, this is a no-brainer correction.

6) The similar Append does not have a Count parameter in any of its declarations. It seems pretty clear that this is a cut-and-paste error.

7) Unfortunately, Ada doesn't provide a mechanism to ensure that a component like Storage is aligned properly. (We don't care *where* it's stored, only that it is aligned to the maximum supported.) Thus we can't actually fix this code as it stands.

The author of the question suggested a complex rewrite using Integer_Address, but that assumed that Integer_Address was a modular type, which is not required by the Ada Standard. That just swaps one problem for another (somewhat less likely) problem. Thus, we adopt a more minimal fix.

Aside: We really ought to look into some solution for the component alignment problem, as it makes it hard to write portable storage pools.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0127**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **3.10.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0127 Incomplete views and access to class-wide types

Working Reference Number AI12-0137-1

Question

3.10.1(2.2-2.6) says:

Given an access type A whose designated type T is an incomplete view, a dereference of a value of type A also has this incomplete view except when:

* it occurs within the immediate scope of the completion of T, or

* it occurs within the scope of a nonlimited_with_clause that mentions a library package in whose visible part the completion of T is declared, or

* it occurs within the scope of the completion of T and T is an incomplete view declared by an incomplete_type_declaration. In these cases, the dereference has the view of T visible at the point of the dereference.

Does this apply to an access-to-class-wide type? (Yes.) A class-wide type does not have an explicit declaration, an incomplete view, or a completion.

Summary of Response

The rules in 3.10.1(2/2 - 2.6/3) concerning uses of access types that designate incomplete views also apply to access types that designate incomplete views of class-wide types.

Corrigendum Wording

Replace 3.10.1(2.1):

An incomplete_type_declaration declares an *incomplete view* of a type and its first subtype; the first subtype is unconstrained if a discriminant_part appears. If the incomplete_type_declaration includes the reserved word **tagged**, it declares a *tagged incomplete view*. An incomplete view of a type is a limited view of the type (see 7.5).

by:

An incomplete_type_declaration declares an *incomplete view* of a type and its first subtype; the first subtype is unconstrained if a discriminant_part appears. If the incomplete_type_declaration includes the reserved word **tagged**, it declares a *tagged incomplete view*. If *T* denotes a tagged incomplete view, then *T*'Class denotes a tagged incomplete view. An incomplete view of a type is a limited view of the type (see 7.5).

Discussion

The question claims that a class-wide type does not have an incompelete view, because it doesn't have an explicit declaration, and thus neither 3.10.1(2.1/2) nor 10.1.1(12.3/3) [whichever is appropriate for the kind of incomplete view] applies to it.

This is clearly the problem, in that a class-wide type should mirror its associated specific type as closely as possible.

Indeed, there doesn't seem to be any general rule about properties of specific types being associated with the class-wide type. 3.4.1(4-5) talks about components and values. 7.5(5/3) extended limitedness. There may be others.

Thus, it appears that this property needs to be explicitly spelled out in the Standard. As such, we add a sentence in 3.10.1(2.1/2) to do that, and then all is well, as 3.10.1(2.2-2.6/3) then clearly apply.

Alternative approaches of fixing 3.10.1(2.2-2.6/3) to handle this case explicitly always ran into the fatal flaw of there being no incomplete view of T'Class to talk about. Once that is fixed, there is no need to make any changes to 3.10.1(2.2-2.6/3).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0128**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.10.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0128 Accessibility level of explicitly aliased parameters of procedures and entries

Working Reference Number AI12-0067-1

Question

3.10.2(7/3) says in part:

> Other than for an explicitly aliased parameter, a formal parameter of a callable entity has the same accessibility level as the master representing the invocation of the entity.

and later 3.10.2(13.3/3) says:

> The accessibility level of an explicitly aliased (see 6.1) formal parameter in a function body is determined by the point of call; it is the same level that the return object ultimately will have.

There seems to be no mention of the accessibility level of an explicitly aliased parameter which is not a parameter of a function. Should this be fixed? (Hell yes.)

Summary of Response

Only explicitly aliased parameters of functions have special accessibility; explicitly aliased parameters of procedures and entries have the same accessibility as other kinds of parameters.

Corrigendum Wording

Replace 3.10.2(7):

- An entity or view defined by a declaration and created as part of its elaboration has the same accessibility level as the innermost master of the declaration except in the cases of renaming and derived access types described below. Other than for an explicitly aliased parameter, a formal parameter of a callable entity has the same accessibility level as the master representing the invocation of the entity.

by:

- An entity or view defined by a declaration and created as part of its elaboration has the same accessibility level as the innermost master of the declaration except in the cases of renaming and derived access types described below. Other than for an explicitly aliased parameter of a function, a formal parameter of a callable entity has the same accessibility level as the master representing the invocation of the entity.

Response

(See !summary.)

Discussion

The special accessibility rules only apply to explicitly aliased parameters of functions, since the accessibility is that of the function result (which doesn't exist for a procedure or entry, of course). So the exception should be limited to just that case.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0129**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.10.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0129 Accessibility rules need to take into account that a generic function is not a

Working Reference Number AI12-0089-1

Question

Repeat after me: a generic function is not a function. :-)

However, the special accessibility rules for aliased parameters of functions do not take this into account, and thus do not apply inside of a generic function.

Thus, an example like:

```
type Return_It (D : access Element) is null record;

generic
function Foobar (X : aliased in out Bounded_Container; C : in Cursor) return
Return_It;

function Foobar (X : aliased in out Bounded_Container; C : in Cursor) return
Return_It is
begin
    ...
    return R : Return_It (X(C)'access); -- What accessibility check here?
end Foobar;
```

would seem to fail a compile-time accessibility in the return statement, whereas the non-generic equivalent would always be legal.

This is weird, should be it be changed? (Yes.)

Summary of Response

The accessibility rules for explicitly aliased parameters and access results also apply in generic function bodies.

Corrigendum Wording

Replace 3.10.2(7):

- An entity or view defined by a declaration and created as part of its elaboration has the same accessibility level as the innermost master of the declaration except in the cases of renaming and derived access types described below. Other than for an explicitly aliased parameter, a formal parameter of a callable entity has the same accessibility level as the master representing the invocation of the entity.

by:

- An entity or view defined by a declaration and created as part of its elaboration has the same accessibility level as the innermost master of the declaration except in the cases of renaming and derived access types described below. Other than for an explicitly aliased parameter of a function or generic function, a formal parameter of a callable entity has the same accessibility level as the master representing the invocation of the entity.

Replace 3.10.2(19.2):

- Inside a return statement that applies to a function *F*, when determining whether the accessibility level of an explicitly aliased parameter of *F* is statically deeper than the level of the return object of *F*, the level of the return object is considered to be the same as that of the level of the explicitly aliased parameter; for statically comparing with the level of other entities, an explicitly aliased parameter of *F* is considered to have the accessibility level of the body of *F*.

by:

- Inside a return statement that applies to a function or generic function *F*, when determining whether the accessibility level of an explicitly aliased parameter of *F* is statically deeper than the level of the return object of *F*, the level of the return object is considered to be the same as that of the level of the explicitly aliased parameter; for statically comparing with the level of other entities, an explicitly aliased parameter of *F* is considered to have the accessibility level of the body of *F*.

Replace 3.10.2(19.3):

- For determining whether a level is statically deeper than the level of the anonymous access type of an access result of a function, when within a return statement that applies to the function, the level of the master of the call is presumed to be the same as that of the level of the master that elaborated the function body.

by:

- For determining whether a level is statically deeper than the level of the anonymous access type of an access result of a function or generic function *F*, when within a return statement that applies to *F*, the level of the master of the call is presumed to be the same as that of the level of the master that elaborated the body of *F*.

Discussion

Repeat after me: a generic function is not a function, a generic procedure is not a procedure, and a generic package is not a package. Even long-time Ada experts tend to forget this, and we clearly did when we fixed the rules in AI12-0067-1, not to mention when we created the rules in the first place.

This falls under the Dewar rule that the Standard does not say nonsense even when it literally does say nonsense. No one would expect a function and a similar generic function to have different accessibility rules for aliased parameters.

We have to change the rule for access results of a function, as it has the same problem.

Note that we don't have to change any dynamic semantics rules, as those can only occur in an instance of a generic function, which IS a function. [A generic unit is not executable, only an instance is.] This means that we don't need to modify 3.10.2(13.3/3), for example.

There is a similar problem in the Legality Rules of 6.5. However, that is mitigated somewhat as a generic function body is considered a function body, and most of the rules are about function bodies rather than functions per-se. Thus we just add a pair of AARM notes.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0130**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **3.10.2; 4.6; 6.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0130 Access values should never designate unaliased components

Working Reference Number AI12-0027-1

Question

We don't want access values pointing to unaliased objects.

But there appear to be several ways to create such values.

4.6(24.8) prohibits view conversions from array types with unaliased components to array types with aliased components. However, value conversions can also create such components. Consider:

```
type T is tagged null record;

type Rec is record F1 : T; F2 : Integer; end record;

type Rec_Ref is access constant Rec;
Ptr : Rec_Ref;

subtype Index is Integer range 1 .. 3;

type A1 is array (Index) of Rec;
type A2 is array (Index) of aliased Rec;

X1 : A1;
begin
Ptr := A2 (X1)(Index'First)'access;
-- Ptr now designates an unaliased object
```

In addition, the existing wording doesn't deal with contract model issues with generics. By 12.5.3(8) and AARM 12.5.3(8.a), it's OK if a formal array type's components are unaliased and the actual's are aliased, and Legality Rules such as 4.6(24.8) aren't checked in an instance body.

Summary of Response

Composite value conversions can cause a copy of the operand object to be made; as such, the accessibility level of such conversions is usually local.

Corrigendum Wording

Replace 3.10.2(10):

- The accessibility level of an aggregate that is used (in its entirety) to directly initialize part of an object is that of the object being initialized. In other contexts, the accessibility level of an aggregate is that of the innermost master that evaluates the aggregate.

by:

- The accessibility level of an aggregate that is used (in its entirety) to directly initialize part of an object is that of the object being initialized. In other contexts, the accessibility level of an aggregate is that of the innermost master that evaluates the aggregate. Corresponding rules apply to a value conversion (see 4.6).

Replace 4.6(24.17):

- The accessibility level of the operand type shall not be statically deeper than that of the target type, unless the target type is an anonymous access type of a stand-alone object. If the target type is that of such a stand-alone object, the accessibility level of the operand type shall not be statically deeper than that of the declaration of the stand-alone object. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit.

by:

- The accessibility level of the operand type shall not be statically deeper than that of the target type, unless the target type is an anonymous access type of a stand-alone object. If the target type is that of such a stand-alone object, the accessibility level of the operand type shall not be statically deeper than that of the declaration of the stand-alone object.

Replace 4.6(24.21):

- The accessibility level of the operand type shall not be statically deeper than that of the target type. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit. If the operand type is declared within a generic body, the target type shall be declared within the generic body.

by:

- The accessibility level of the operand type shall not be statically deeper than that of the target type. If the operand type is declared within a generic body, the target type shall be declared within the generic body.

In addition to the places where Legality Rules normally apply (see 12.3), these rules apply also in the private part of an instance of a generic unit.

Insert after 4.6(58):

Conversion to a type is the same as conversion to an unconstrained subtype of the type.

the new paragraphs:

Evaluation of a value conversion of a composite type either creates a new anonymous object (similar to the object created by the evaluation of an aggregate or a function call) or yields a new view of the operand object without creating a new object:

- If the target type is a by-reference type and there is a type that is an ancestor of both the target type and the operand type then no new object is created;
- If the target type is an array type having aliased components and the operand type is an array type having unaliased components, then a new object is created;
- Otherwise, it is unspecified whether a new object is created.

If a new object is created, then the initialization of that object is an assignment operation.

Replace 6.2(10):

A parameter of a by-reference type is passed by reference, as is an explicitly aliased parameter of any type. Each value of a by-reference type has an associated object. For a parenthesized expression, qualified_expression, or type_conversion, this object is the one associated with the operand. For a conditional_expression, this object is the one associated with the evaluated *dependent*_expression.

by:

A parameter of a by-reference type is passed by reference, as is an explicitly aliased parameter of any type. Each value of a by-reference type has an associated object. For a parenthesized expression, qualified_expression, or view conversion, this object is the one associated with the operand. For a value conversion, the associated object is the anonymous result object if such an object is created (see 4.6); otherwise it is the associated object of the operand. For a conditional_expression, this object is the one associated with the evaluated *dependent*_expression.

Discussion

Ada 2012 never defines the accessibility of value conversions. However, these are expected to be able to change representations (as described in 13.6), and such a change surely will require a copy. If the type contains aliased components, surely those will need to be copied, too.

Thus, simply designing a rule to make cases like the one in the question directly illegal doesn't fix anything. Moreover, as it needs to be enforced in an assume-the-worst manner in generic bodies, it could make existing code that has no problem illegal. That doesn't seem very promising.

Rather, we explicitly define the copying done by a value conversion, and define the accessibility of such a conversion when it is copied to be similar to that for an aggregate defined in the same place. That will give such components a very short lifetime such that most attempts to take 'Access of them will fail an accessibility check. As most such checks are made statically, this will not add much overhead.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0131**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **3.10.2; 9.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0131 9.3(2) does not work for anonymous access types

Working Reference Number AI12-0070-1

Question

9.3(2) says:

> If the task is created by the evaluation of an allocator for a given access type, it depends on each master that includes the elaboration of the declaration of the ultimate ancestor of the given access type.

This rule hasn't changed since the original Ada 95. This rule was written when all access types were named, and it works fine when an allocator type is a named access type. But if the type of an allocator is an anonymous access type, this rule doesn't make any sense. Anonymous access types don't have declarations of their own; they are part of some larger declaration. Moreover, we have lots of rules that specify what the accessibility (and thus the master) of an anonymous access type is. This rule overrides all of that, and would cause bizarre effects for access results, stand-alone objects of an anonymous access type, and access discriminants. Is this intended? (Surely not.)

Summary of Response

The master of an anonymous access type is defined in 7.6.1 (often via 3.10.2), and a task created by an allocator for an anonymous access type has that master, regardless of whatever 9.3 says.

Corrigendum Wording

Insert after 3.10.2(13.1):

- The accessibility level of the anonymous access type of an access parameter specifying an access-to-subprogram type is deeper than that of any master; all such anonymous access types have this same level.

the new paragraph:

- The accessibility level of the anonymous access subtype defined by a return_subtype_indication that is an access_definition (see 6.5) is that of the result subtype of the enclosing function.

Replace 9.3(2):

- If the task is created by the evaluation of an allocator for a given access type, it depends on each master that includes the elaboration of the declaration of the ultimate ancestor of the given access type.

by:

- If the task is created by the evaluation of an allocator for a given named access type, it depends on each master that includes the elaboration of the declaration of the ultimate ancestor of the given access type.

Response

(See !summary.)

Discussion

3.10.2(14/3) and its many related paragraphs clearly define the accessibility level of an object declared by an allocator; and each accessibility level has an associated master. It's clearly the intent that these (complex) rules apply to tasks as well as finalization; there were many examples of task termination discussed during the crafting of these rules.

So 9.3(2) must at least refer to those rules and not try to invent its own.

We chose the most minimum possible fix, just changing the existing bullet to only apply to named access types. This depends on the fact that 9.2(3.1/2) is sufficient to describe the behavior for ALL cases; as such, we use it to describe the behavior for all but the most basic cases.

This also fixes a related problem. A stand-alone object of an anonymous access type (SAOAAT) is an object_declaration. So for something like: A : access A_Task_Type := new A_Task_Type; both the original 9.3(2) and 9.3(3) apply! With the fix, 9.3(2) clearly no longer applies.

A vaguely related point:

The questioner also notes that 6.5(5.3) is the only rule that defines the accessibility of something that is not in 3.10.2. Thus, a copy of it is added to 3.10.2 for completeness, and the original was marked redundant.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0132**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.10.2; 4.3.3; 6.8; 7.5; 13.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0132 Missing rules for expression functions

Working Reference Number AI12-0157-1

Question

(1) If an expression function needs to return an aggregate, the syntax requires double parens:

```
function Rounds return VExt is ((F732C00.Rounds with H => CNPriv));
```

as the syntax is

[overriding_indicator] function_specification is (expression) [aspect_specification];

and an aggregate is also surrounded by parentheses.

This is ugly, and inconsistent with the rest of Ada. In particular, it is inconsistent with the behavior of aggregates in qualified expressions and allocators, and the behavior of other things surrounded in parentheses like conditional expressions and raise expressions.

Should aggregates be allowed here? (Yes.)

(2) 4.3.3(11/2) defines the applicable index constraint of an aggregate in many contexts. We surely want the return version to apply to an expression function as well, but it's not clear that the wording has that effect. We define an expression function dynamically to be equivalent to the execution of a simple_return_statement, but we treat the legality and static semantic rules explicitly in 6.8. 4.3.3(11/2) is a Legality Rule, thus it arguably is not covered by 6.8(7/3). Should it be updated to include expression functions? (Yes.)

(3) 3.10.2(19.2/4) and 3.10.2(19.3/4) define the statically deeper relationship for return statements inside a function. The same rules need to apply within an expression function. (Note that we don't need to make any changes to the dynamic accessibility level, such as 3.10.2(10.8/3), since 6.8(7/3) clearly does apply to runtime effects.) Should this be fixed? (Yes.)

Summary of Response

(1) Expression functions can directly use an aggregate as well as an expression surrounded by parentheses.

(2) The expression or aggregate of an expression function has a defined applicable index constraint.

(3) The statically deeper relationship is defined as for a return statement for the expression or aggregate of an expression function.

Corrigendum Wording

Replace 3.10.2(19.2):

- Inside a return statement that applies to a function or generic function *F*, when determining whether the accessibility level of an explicitly aliased parameter of *F* is statically deeper than the level of the return object of *F*, the level of the return object is considered to be the same as that of the level of the explicitly aliased parameter; for statically comparing with the level of other entities, an explicitly aliased parameter of *F* is considered to have the accessibility level of the body of *F*.

by:

- Inside a return statement that applies to a function or generic function *F*, or the return expression of an expression function *F*, when determining whether the accessibility level of an explicitly aliased parameter of *F* is statically deeper than the level of the return object of *F*, the level of the return object is considered to be the same as that of the level of the explicitly aliased parameter; for statically comparing with the level of other entities, an explicitly aliased parameter of *F* is considered to have the accessibility level of the body of *F*.

Replace 3.10.2(19.3):

- For determining whether a level is statically deeper than the level of the anonymous access type of an access result of a function or generic function *F*, when within a return statement that applies to *F*, the level of the master of the call is presumed to be the same as that of the level of the master that elaborated the body of *F*.

by:

- For determining whether a level is statically deeper than the level of the anonymous access type of an access result of a function or generic function *F*, when within a return statement that applies to *F* or the return expression of expression function *F*, the level of the master of the call is presumed to be the same as that of the level of the master that elaborated the body of *F*.

Replace 4.3.3(11):

For an explicit_actual_parameter, an explicit_generic_actual_parameter, the expression of a return statement, the initialization expression in an object_declaration, or a default_expression (for a parameter or a component), when the nominal subtype of the corresponding formal parameter, generic formal parameter, function return object, object, or component is a constrained array subtype, the applicable index constraint is the constraint of the subtype;

by:

For an explicit_actual_parameter, an explicit_generic_actual_parameter, the expression of a return statement, the return expression of an expression function, the initialization expression in an object_declaration, or a default_expression (for a parameter or a component), when the nominal subtype of the corresponding formal parameter, generic formal parameter, function return object, expression function return object, object, or component is a constrained array subtype, the applicable index constraint is the constraint of the subtype;

Replace 6.8(2):

```
expression_function_declaration ::=
   [overriding_indicator]
   function_specification is
      (expression)
      [aspect_specification];
```

by:

```
expression_function_declaration ::=
   [overriding_indicator]
   function_specification is
      (expression)
      [aspect_specification];
 | [overriding_indicator]
   function_specification is
      aggregate
      [aspect_specification];
```

Replace 6.8(3):

The expected type for the expression of an expression_function_declaration is the result type (see 6.5) of the function.

by:

The expected type for the expression or aggregate of an expression_function_declaration is the result type (see 6.5) of the function.

Replace 6.8(5):

If the result subtype has one or more unconstrained access discriminants, the accessibility level of the anonymous access type of each access discriminant, as determined by the expression of the expression function, shall not be statically deeper than that of the master that elaborated the expression_function_declaration.

by:

If the result subtype has one or more unconstrained access discriminants, the accessibility level of the anonymous access type of each access discriminant, as determined by the expression or aggregate of

the expression_function_declaration, shall not be statically deeper than that of the master that elaborated the expression_function_declaration.

Replace 6.8(6):

An expression_function_declaration declares an *expression function*. A completion is not allowed for an expression_function_declaration; however, an expression_function_declaration can complete a previous declaration.

by:

An expression_function_declaration declares an *expression function*. The *return expression* of an expression function is the expression or aggregate of the expression_function_declaration. A completion is not allowed for an expression_function_declaration; however, an expression_function_declaration can complete a previous declaration.

Replace 6.8(7):

The execution of an expression function is invoked by a subprogram call. For the execution of a subprogram call on an expression function, the execution of the subprogram_body executes an implicit function body containing only a simple_return_statement whose expression is that of the expression function.

by:

The execution of an expression function is invoked by a subprogram call. For the execution of a subprogram call on an expression function, the execution of the subprogram_body executes an implicit function body containing only a simple_return_statement whose expression is the return expression of the expression function.

Replace 7.5(2.9):

- the expression of an expression_function_declaration (see 6.8)

by:

- the return expression of an expression function (see 6.8)

Replace 13.14(5.1):

- At the occurrence of an expression_function_declaration that is a completion, the expression of the expression function causes freezing.

by:

- At the occurrence of an expression_function_declaration that is a completion, the return expression of the expression function causes freezing.

Replace 13.14(5.2):

- At the occurrence of a renames-as-body whose *callable_entity*_name denotes an expression function, the expression of the expression function causes freezing.

by:

- At the occurrence of a renames-as-body whose *callable_entity*_name denotes an expression function, the return expression of the expression function causes freezing.

Replace 13.14(8):

A static expression (other than within an aspect_specification) causes freezing where it occurs. An object name or nonstatic expression causes freezing where it occurs, unless the name or expression is part of a default_expression, a default_name, the expression of an expression function, an aspect_specification, or a per-object expression of a component's constraint, in which case, the freezing occurs later as part of another construct or at the freezing point of an associated entity.

by:

A static expression (other than within an aspect_specification) causes freezing where it occurs. An object name or nonstatic expression causes freezing where it occurs, unless the name or expression is part of a default_expression, a default_name, the return expression of an expression function, an

aspect_specification, or a per-object expression of a component's constraint, in which case, the freezing occurs later as part of another construct or at the freezing point of an associated entity.

Replace 13.14(10.1):

- At the place where a function call causes freezing, the profile of the function is frozen. Furthermore, if a parameter of the call is defaulted, the default_expression for that parameter causes freezing. If the function call is to an expression function, the expression of the expression function causes freezing.

by:

- At the place where a function call causes freezing, the profile of the function is frozen. Furthermore, if a parameter of the call is defaulted, the default_expression for that parameter causes freezing. If the function call is to an expression function, the return expression of the expression function causes freezing.

Replace 13.14(10.2):

- At the place where a generic_instantiation causes freezing of a callable entity, the profile of that entity is frozen unless the formal subprogram corresponding to the callable entity has a parameter or result of a formal untagged incomplete type; if the callable entity is an expression function, the expression of the expression function causes freezing.

by:

- At the place where a generic_instantiation causes freezing of a callable entity, the profile of that entity is frozen unless the formal subprogram corresponding to the callable entity has a parameter or result of a formal untagged incomplete type; if the callable entity is an expression function, the return expression of the expression function causes freezing.

Replace 13.14(10.3):

- At the place where a use of the Access or Unchecked_Access attribute whose prefix denotes an expression function causes freezing, the expression of the expression function causes freezing.

by:

- At the place where a use of the Access or Unchecked_Access attribute whose prefix denotes an expression function causes freezing, the return expression of the expression function causes freezing.

Discussion

(1) Consistency with qualified expressions seems important here. An early implementation of Ada 2012 allows this syntax (presumably by accident), and an ACATS test-writer used it (definitely by accident, known because the author of this AI and that test-writer are one and the same). It seems natural, and it's best if the syntax of Ada follows that as much as possible.

(2) It appears that a strict reading of 4.3.3 says that

```
function Cool return Some_Array is ((others => 1));
```

or, with adoption of (1)

```
function Cool return Some_Array is (others => 1);
```

are illegal because an "applicable index constraint" isn't provided (this is none of the contexts listed in 4.3.3(11-15.1)). We don't want that, since the equivalent

```
function Cool return Some_Array is
begin
    return (others => 1);
end Cool;
```

is legal.

(3) This was raised by Steve Baird (who else?). It is unlikely but possible for these 3.10.2 rules to matter. They would matter in a case like:

```
function Bar (Obj   : aliased in out Int_Cont;
              Index : in Natural) return access Integer is
   (Obj.Element(Index)'access);
```

We want this to be legal, as it is legal if this is written with an explicit body. (Bar here is similar to the construction of function Reference in the containers; we would want it to be possible for an expression function to be used in place of a full body for these routines.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0133**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.10.2; 6.4.1; 12.5.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0133 Generic formal types and constrained partial views

Working Reference Number AI12-0095-1

Question

6.4.1(6.2/3) says:

If the formal parameter is an explicitly aliased parameter, the type of the actual parameter shall be tagged or the actual parameter shall be an aliased view of an object. Further, if the formal parameter subtype F is untagged:

* the subtype F shall statically match the nominal subtype of the actual object; or

* the subtype F shall be unconstrained, discriminated in its full view, and unconstrained in any partial view.

Consider applying this rule in a generic body:

```
generic
   type P (D : Natural) is private;
   Obj : P;
package G is
   procedure Proc (Param : aliased in P);
end G;

package body G is
   procedure Proc (Param : aliased in P) is ...

begin
    declare
       O1 : aliased P(1) := Obj;
       type AP is access all P;
       OAP : AP;
    begin
       Proc (O1); -- Legal?
       OAP := O1'access; -- Not legal.
    end;
end G;
```

(Ignore the possibility of the declaration of O1 raising Constraint_Error for the moment; that has no effect on Legality.)

O1'access is not legal by 3.10.2(27.2/3), which says:

> D shall be discriminated in its full view and unconstrained in any partial view, and the designated subtype of A shall be unconstrained. For the purposes of determining within a generic body whether D is unconstrained in any partial view, a discriminated subtype is considered to have a constrained partial view if it is a descendant of an untagged generic formal private or derived type.

The second sentence is an "assume-the-worst" rule for generic bodies, because of which the nominal subtype has to statically match in such cases.

The rules in 6.4.1(6-6.2/3) are intended to be similar, so that an aliased parameter doesn't provide a way to "launder" an object and allow it to be the prefix of 'Access when the object directly would not allow that. But for some reason, the second sentence of 3.10.2(27.2/3) didn't make it into 6.4.1(6.2/3). Perhaps that's because it was added by AI05-0041-1, so maybe it appeared later than the 6.4.1 wording.

Should this be fixed? (Yes.)

Summary of Response

Within a generic body, we assume the worst as to whether or not a formal subtype has a constrained partial view. In particular, we assume that untagged formal private and derived types have such a view.

Corrigendum Wording

Replace 3.10.2(27.2):

- *D* shall be discriminated in its full view and unconstrained in any partial view, and the designated subtype of *A* shall be unconstrained. For the purposes of determining within a generic body whether *D* is unconstrained in any partial view, a discriminated subtype is considered to have a constrained partial view if it is a descendant of an untagged generic formal private or derived type.

by:

- *D* shall be discriminated in its full view and unconstrained in any partial view, and the designated subtype of *A* shall be unconstrained.

Insert after 6.4.1(6.2):

- the subtype *F* shall be unconstrained, discriminated in its full view, and unconstrained in any partial view.

the new paragraph:

In addition to the places where Legality Rules normally apply (see 12.3), these rules also apply in the private part of an instance of a generic unit.

Insert after 12.5.1(15):

For a generic formal type with an unknown_discriminant_part, the actual may, but need not, have discriminants, and may be definite or indefinite.

the new paragraph:

When enforcing Legality Rules, for the purposes of determining within a generic body whether a type is unconstrained in any partial view, a discriminated subtype is considered to have a constrained partial view if it is a descendant of an untagged generic formal private or derived type.

Discussion

It was pointed out that there is a similar rule for access type conversions in 4.6(24.16/3). This exists for the same reason - an access type conversion should not be able to "launder" an access value that could not be created directly with 'access.

Since this rule occurs (at least) 3 times in the Standard, we add a blanket rule to 12.5.1 rather than duplicating the wording in all three places. This should avoid future maintenance mistakes.

We add the generic boilerplate to 6.4.1(6.2/3). It already applies to all 3.10.2 Legality Rules for the access attribute, and it was added by AI12-0027-1 to all type conversion rules. As noted above, we want these all to work the same way.

Note that the author had (half-heartedly) proposed in AI05-0041-1 that we define the term "known to be unconstrained" for this purpose. (This would be defined as the entire 3.10.2(27.2/3) bullet.) That would be an alternative to the chosen solution, but it was rejected as it would potentially be confused with "known to be constrained". [A detailed explanation of this solution can be found in the !appendix.]

This change to the rules is incompatible for the type conversion and aliased parameter cases. In both cases, this is similar to the incompatibility introduced for the prefix of 'Access by AI05-0041-1, except that it is even less likely in these cases. For aliased parameters (new to Ada 2012), there probably isn't enough use of them for the incompatibility to be significant. For type conversions (the rule in its current form was introduced in Ada 2005), the likelihood of problems is low, as a generic formal private or derived type with discriminants is required, along with a type conversion from one access type to another with the same designated type that does not statically match.

While working on this AI, we also noticed that the AARM Note 6.4.1(6.d/3) is wrong, as "master of the function call" defines other cases where the master of the function (same as master of the return object) is not

the same as the master of the execution of the call. This note probably predates the definition of "master of the function call" by AI05-0234-1; we've also corrected it.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0134**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **4.1.4; 6.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0134 Questions on 'Old

Working Reference Number AI12-0032-1

Question

1) X'Old denotes a (constant) object. The accessibility level of that object does not seem to be defined by the current wording of the Standard. Should it be? (It's OK as is.)

2) X and X'Old are defined to have the same type, even if that type is anonymous.

That's usually good. We want to allow:

Table : array (1 .. 3) of Integer;

```
procedure Foo
  with Post (Table = Table'Old);
```

If, however, X is of an anonymous access type, then things get odd. What does it mean to say that a standalone constant has the same type as, say, an access discriminant? This makes no sense. Even if X itself is a standalone object, having two such objects sharing the same type leads to problems (for example, does the accessibility level of the type of X'Old then depend on the value most recently assigned to X?).

Should we define the type of the constant to be an anonymous access with the same designated type? (Yes.)

3) Should the nominal subtype of X'Old be defined? (Yes and it is.) It currently isn't, meaning we fall back to 4.1.4(9/3), which says it is the base subtype of the type.

That means that the following is illegal:

```
procedure Old_Test is

  procedure Foo (X : in out Natural)
    with Post => X = (case X'Old is when 0 => 0,
                                     when Positive => X'Old - 1);

  procedure Foo (X : in out Natural) is
   begin if X > 0 then X := X - 1; end if; end;

begin null; end;

because the case expression doesn't cover all values of the subtype of the
case expression.
```

4) What should happen if X'Old has an abstract type? (X'Old has the same runtime tag as X.) A direct application of the model would imply that it should be illegal, because we'd be declaring a constant of an abstract type. But it seems odd that X'Old should be illegal even if X is legal.

5) If T is a specific tagged type, the runtime tags of X and X'Old may not match, which can affect (among other things) the body executed by a dispatching call. That's because the implicit constant declaration of: X'Old : constant T := X; always has the tag of T, while X may have some other tag because of type conversions. This affects (re)dispatching and equality, among other things. Should this be fixed somehow? (Yes.)

6) What is the meaning of Named_Number'Old? (See summary.) This is something that can't easily be declared (there are no objects of a universal type), and it's useless as a named number has to be constant anyway.

7) An occurrence of X'Old in a postcondition is evaluated before the associated implicitly declared constant is finalized. In the case of an entry or protected subprogram, does this mean that postcondition checking is peformed within the rendezvous or protected action? (Yes.)

Summary of Response

(1) The accessibility level of X'Old is that of the associated implied constant declaration.

(2) If X has an anonymous access type, the implied constant associated with X'Old has an anonymous access type with the same designated type or profile.

(3) The nominal subtype of X'Old is defined to be that of X.

(4) X'Old is legal for an abstract type, as X'Old has the same underlying runtime tag as X (which necessarily cannot be abstract).

(5) If X is tagged, X'Old has the same underlying runtime tag as X.

(6) X'Old is well defined if X is of a universal type.

(7) In the case of an entry or a protected subprogram with a postcondition the events listed below occur in the following order: - any postcondition checks are performed; then - any constants associated with 'Old occurrences are finalized; then - the rendezvous or protected action is ended.

Corrigendum Wording

Replace 4.1.4(9):

> An attribute_reference denotes a value, an object, a subprogram, or some other kind of program entity. For an attribute_reference that denotes a value or an object, if its type is scalar, then its nominal subtype is the base subtype of the type; if its type is tagged, its nominal subtype is the first subtype of the type; otherwise, its nominal subtype is a subtype of the type without any constraint or null_exclusion. Similarly, unless explicitly specified otherwise, for an attribute_reference that denotes a function, when its result type is scalar, its result subtype is the base subtype of the type, when its result type is tagged, the result subtype is the first subtype of the type, and when the result type is some other type, the result subtype is a subtype of the type without any constraint or null_exclusion.

by:

> An attribute_reference denotes a value, an object, a subprogram, or some other kind of program entity. Unless explicitly specified otherwise, for an attribute_reference that denotes a value or an object, if its type is scalar, then its nominal subtype is the base subtype of the type; if its type is tagged, its nominal subtype is the first subtype of the type; otherwise, its nominal subtype is a subtype of the type without any constraint or null_exclusion. Similarly, unless explicitly specified otherwise, for an attribute_reference that denotes a function, when its result type is scalar, its result subtype is the base subtype of the type, when its result type is tagged, the result subtype is the first subtype of the type, and when the result type is some other type, the result subtype is a subtype of the type without any constraint or null_exclusion.

Insert after 6.1.1(22):

- a *dependent_*expression of a case_expression;

the new paragraph:

- the predicate of a quantified_expression;

Replace 6.1.1(26):

> X'Old
>
> For each X'Old in a postcondition expression that is enabled, a constant is implicitly declared at the beginning of the subprogram or entry. The constant is of the type of X and is initialized to the result of evaluating X (as an expression) at the point of the constant declaration. The value of X'Old in the postcondition expression is the value of this constant; the type of X'Old is the type of X. These implicit constant declarations occur in an arbitrary order.

by:

X'Old

Each X'Old in a postcondition expression that is enabled denotes a constant that is implicitly declared at the beginning of the subprogram body, entry body, or accept statement.

The implicitly declared entity denoted by each occurrence of X'Old is declared as follows:

- If X is of an anonymous access defined by an access_definition *A* then

  ```
  X'Old : constant A := X;
  ```

- If X is of a specific tagged type *T* then

  ```
  anonymous : constant T'Class := T'Class (X);
  X'Old : T renames T(anonymous);
  ```

  where the name X'Old denotes the object renaming.

- Otherwise

  ```
  X'Old : constant S := X;
  ```

  where *S* is the nominal subtype of X. This includes the case where the type of *S* is an anonymous array type or a universal type.

The nominal subtype of X'Old is as implied by the above definitions. The expected type of the prefix of an Old attribute is that of the attribute. Similarly, if an Old attribute shall resolve to be of some type, then the prefix of the attribute shall resolve to be of that type.

Insert after 6.1.1(35):

The precondition checks are performed in an arbitrary order, and if any of the class-wide precondition expressions evaluate to True, it is not specified whether the other class-wide precondition expressions are evaluated. The precondition checks and any check for elaboration of the subprogram body are performed in an arbitrary order. It is not specified whether in a call on a protected operation, the checks are performed before or after starting the protected action. For an entry call, the checks are performed prior to checking whether the entry is open.

the new paragraph:

For a call to a task entry, the postcondition check is performed before the end of the rendezvous; for a call to a protected operation, the postcondition check is performed before the end of the protected action of the call. The postcondition check for any call is performed before the finalization of any implicitly-declared constants associated (as described above) with Old attribute_references but after the finalization of any other entities whose accessibility level is that of the execution of the callable construct.

Discussion

'Old constants are declared "at the beginning of the subprogram body, entry body, or accept statement". This means that, in the case of an entry or a protected subprogram, they are declared after the start of the associated rendezvous or protected action. Since such constants can only be usefully referenced before they are finalized, postcondition checks (which need to be able to reference these constants) must be performed before, not after, the end of the rendezvous or protected action.

We add quantified_expressions to expressions that are potentially unevaluated, because the compiler cannot know how many times (if at all) the predicate is evaluated. 6.1.1(27/3) makes (for I in 1 .. 10 => A(I)'Old = A(I)) illegal, but we also need (for I in 1 .. F(...) => A(F(N)'Old)) to be illegal.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0135**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.1.5; 4.1.6; 5.5.1; 13.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0135 Iterators of formal derived types

Working Reference Number AI12-0138-1

Question

Consider a root type T that has a reversible iterator, and a derivation NT of that type that has only a forward iterator. Then for a generic generic type FT is new T with private; package G is ...

In the generic body we could write an iterator using "reverse" for type FT. We would want an instance using NT to fail (since there is no reverse available), but of course there is no recheck in a generic body.

There is a contract problem here.

Summary of Response

We define the notion of "nonoverridable" aspects, and declare Default_Iterator, Iterator_Element, Implicit_Dereference, Constant_Indexing, and Variable_Indexing to be nonoverridable aspects. The values of these aspects are names that must not be changed for a derived type (their value can be confirmed), though in each type it is expected that the name might denote different subprograms, types, or discriminants.

Corrigendum Wording

Insert before 4.1.5(6):

A generalized_reference denotes a view equivalent to that of a dereference of the reference discriminant of the reference object.

the new paragraph:

The Implicit_Dereference aspect is nonoverridable (see 13.1.1).

Insert after 4.1.6(5):

An *indexable container type* is (a view of) a tagged type with at least one of the aspects Constant_Indexing or Variable_Indexing specified. An *indexable container object* is an object of an indexable container type. A generalized_indexing is a name that denotes the result of calling a function named by a Constant_Indexing or Variable_Indexing aspect.

the new paragraph:

The Constant_Indexing and Variable_Indexing aspects are nonoverridable (see 13.1.1).

Delete 4.1.6(6):

The Constant_Indexing or Variable_Indexing aspect shall not be specified:

Delete 4.1.6(7):

- on a derived type if the parent type has the corresponding aspect specified or inherited; or

Delete 4.1.6(8):

- on a full_type_declaration if the type has a tagged partial view.

Delete 4.1.6(9):

In addition to the places where Legality Rules normally apply (see 12.3), these rules apply also in the private part of an instance of a generic unit.

Insert after 5.5.1(11):

An *iterable container type* is an indexable container type with specified Default_Iterator and Iterator_Element aspects. A *reversible iterable container type* is an iterable container type with the default iterator type being a reversible iterator type. An *iterable container object* is an object of an iterable container type. A *reversible iterable container object* is an object of a reversible iterable container type.

the new paragraph:

The Default_Iterator and Iterator_Element aspects are nonoverridable (see 13.1.1).

Insert after 13.1.1(18):

A language-defined aspect shall not be specified in an aspect_specification given on a subprogram_body or subprogram_body_stub that is a completion of another declaration.

the new paragraphs:

If an aspect of a derived type is inherited from an ancestor type and has the boolean value True, the inherited value shall not be overridden to have the value False for the derived type, unless otherwise specified in this International Standard.

Certain type-related aspects are defined to be *nonoverridable*; all such aspects are specified using an aspect_definition that is a name.

If a nonoverridable aspect is directly specified for a type *T*, then any explicit specification of that aspect for any other descendant of *T* shall be *confirming*; that is, the specified name shall *match* the inherited aspect, meaning that the specified name shall denote the same declarations as would the inherited name.

If a full type has a partial view, and a given nonoverridable aspect is allowed for both the full view and the partial view, then the given aspect for the partial view and the full view shall be the same: the aspect shall be directly specified only on the partial view; if the full type inherits the aspect, then a matching definition shall be specified (directly or by inheritance) for the partial view.

In addition to the places where Legality Rules normally apply (see 12.3), these rules about nonoverridable aspects also apply in the private part of an instance of a generic unit.

The Default_Iterator, Iterator_Element, Implicit_Dereference, Constant_Indexing, and Variable_Indexing aspects are nonoverridable.

Delete 13.1.1(34):

If an aspect of a derived type is inherited from an ancestor type and has the boolean value True, the inherited value shall not be overridden to have the value False for the derived type, unless otherwise specified in this International Standard.

Discussion

In 4.1.5, no need to eliminate "if not overridden" wording. It is fine as is because of the possibility of a confirming aspect specification.

Similarly, the note in 4.1.6 The Constant_Indexing and Variable_Indexing aspects cannot be redefined when inherited for a derived type, but the functions that they denote can be modified by overriding or overloading. is fine as it stands.

The rules about partial views in the definition of nonoverridable aspects are needed so that privacy is not compromised.

The first new rule on partial views (which was originally in 4.1.6) is needed to prevent problems like this one:

```
package Pkg1 is
   type Rec is record Int : Integer; end record;
   R1, R2 : aliased Rec;

   type T1 (D1, D2 : access Rec) is private;

   generic
      type Descendant is new T1;
   package G is
      X : Descendant (R1'access, R2'access);
      function F return Integer;
   end;
private
   type T1 (D1, D2 : access Rec) is null record with
      Implicit_Dereference => D1; -- Illegal by new rule.
end;
```

```
package body Pkg1 is
   package body G is
      function F return Integer is (X.Int);
   end G;
end;

with Pkg1;
package Pkg2 is
   use Pkg1;
   type T2 is new T1 with Implicit_Dereference => D2; -- No check here.
   package I is new G (T2); -- Trouble here if T1 was legal.
end;
```

The second new partial-view rule is needed to prevent hidden inheritance.

```
package Pkg3 is
   type Parent is tagged null record;
   type Child (D : access Integer) is new Parent with null record
      with Implicit_Dereference => D;
end Pkg3;

with Pkg3;
package Pkg4 is
   type Priv is new Pkg3.Parent with private;
private
   type Priv is new Pkg3.Child with null record;
      -- Illegal by new rule: Priv would have hidden Implicit_Dereference.
end Pkg4;

with Pkg1;
package Pkg4.Child is
   use Pkg4;
   type T3 (D2 : access Integer) is new Priv
      with Implicit_Dereference => D2; -- No check possible here.
end Pkg4.Child;

type T3 would have two Implicit_Dereferences, both visible in the body of
Pkg4.Child, for different discriminants.
```

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0136**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **4.1.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0136 Overriding an aspect is undefined

Working Reference Number AI12-0104-1

Question

4.1.6(4/3) uses some terminology that's not defined:

> These aspects are inherited by descendants of T (including the class-wide type T'Class). The aspects shall not be overridden, but the functions they denote may be.

There is no such thing as overriding an aspect. It's probable that this was intended informally, but then we have an informal use and a formal use of the same term in the same sentence (the formal use is implicit in this wording, but it surely is there).

Should this be changed? (Yes.)

Summary of Response

A confusing sentence regarding changing aspects is replaced by a user note.

Corrigendum Wording

Replace 4.1.6(4):

> These aspects are inherited by descendants of *T* (including the class-wide type *T*'Class). The aspects shall not be overridden, but the functions they denote may be.

by:

> These aspects are inherited by descendants of *T* (including the class-wide type *T*'Class).

Insert after 4.1.6(17):

> When a generalized_indexing is interpreted as a constant (or variable) indexing, it is equivalent to a call on a prefixed view of one of the functions named by the Constant_Indexing (or Variable_Indexing) aspect of the type of the *indexable_container_object*_prefix with the given actual_parameter_part, and with the *indexable_container_object*_prefix as the prefix of the prefixed view.

the new paragraph:

> NOTES
>
> 6 The Constant_Indexing and Variable_Indexing aspects cannot be redefined when inherited for a derived type, but the functions that they denote can be modified by overriding or overloading.

Discussion

The original text here is marked as redundant; the actual rule is given in 4.1.6(7/3). As such, it's obvious that this was intended to be explanatory text. But, as it's given in the normative rules, it's easy to wonder what it means formally. Moreover, as it is a forward reference, a reader may not realize that it isn't normative and try to figure out what rules it implies. By replacing it by a user note, we eliminate any confusion, and we can clarify the sentence so we're not mixing formal and informal uses of the same term.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0137**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **4.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0137 Enforcing legality for anonymous access components in record aggregates

Working Reference Number AI12-0046-1

Question

4.3.1(16/3) says: If a record_component_association with an expression has two or more associated components, all of them shall be of the same type, or all of them shall be of anonymous access types whose subtypes statically match.

There are currently three known situations where this allows constructs which would be legal for some but not all of the associated components.

1)

```
type Rec (D : access Integer) is
   record F : access Integer; end record;

begin
   declare
     X : aliased Integer;
     R : Rec := (D | F => X'access); -- ok for D, not for F
```

2)

```
type T1 is tagged record F1 : access Integer; end record;

begin
   declare
      type T2 is new T1 with record F2 : access Integer; end record;
       X : aliased Integer;
       T2_Obj : T2 :=
         (F1 | F2 => X'access); -- ok for F2, not for F1
```

3) We have T1 and T2 as in example #2 except that both are declared at library level. Exactly one of the two is declared in a unit which has a "null" default storage pool.

> Then the aggregate is (F1 | F2 => new Integer) which is legal only for the component with the non-null default storage pool.

Clearly, any legality rules that are to be applied to a component expression in a record aggregate are to be applied for each associated component. Do we need to state this explicitly? (Yes.)

Summary of Response

The legality of an expression is enforced separately for each associated component.

Corrigendum Wording

Replace 4.3.1(16):

> Each record_component_association other than an **others** choice with a <> shall have at least one associated component, and each needed component shall be associated with exactly one record_component_association. If a record_component_association with an expression has two or more associated components, all of them shall be of the same type, or all of them shall be of anonymous access types whose subtypes statically match.

by:

> Each record_component_association other than an **others** choice with a <> shall have at least one associated component, and each needed component shall be associated with exactly one record_component_association. If a record_component_association with an expression has two

or more associated components, all of them shall be of the same type, or all of them shall be of anonymous access types whose subtypes statically match. In addition, Legality Rules are enforced separately for each associated component.

Discussion

We considered tightening up the rules so that only components with the same legality would be allowed. For instance, consider: If a record_component_association with an expression has two or more associated components, all of them shall be of the same type, or all of them shall be of anonymous access types whose subtypes statically match and whose definitions all occur within one known_discriminant_part or record_definition.

However, this is incompatible with Ada 2005 and Ada 2012. The above would make the following legal Ada 2005 code illegal: type T1 is tagged record F1 : access Integer; end record; type T2 is new T1 with record F2 : access Integer; end record; X : aliased Integer; T2_Obj : T2 := (others => X'Access); -- Ok in Ada 2005, illegal by proposed rule

This problem isn't severe enough to introduce an incompatibility, so this option was dropped.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0138**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.3.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0138 Box expressions in array aggregates

Working Reference Number AI12-0084-1

Question

For box expressions in an array aggregate, 4.3.3(23.1/2) says

> For an array_component_association with <>, the associated component(s) are initialized by default as for a stand-alone object of the component subtype (see 3.3.1).

In the case where the Default_Component_Value aspect of the array type has been specified, this rule ignores it.

This seems wrong. Should Default_Component_Value be taken into account? (Yes.)

Summary of Response

The value of the Default_Component_Value aspect is used for the value of <> in an array aggregate, if it was specified for the array type.

Corrigendum Wording

Replace 4.3.3(23.1):

> Each expression in an array_component_association defines the value for the associated component(s). For an array_component_association with <>, the associated component(s) are initialized by default as for a stand-alone object of the component subtype (see 3.3.1).

by:

> Each expression in an array_component_association defines the value for the associated component(s). For an array_component_association with <>, the associated component(s) are initialized to the Default_Component_Value of the array type if this aspect has been specified for the array type; otherwise, they are initialized by default as for a stand-alone object of the component subtype (see 3.3.1).

Discussion

It would be silly to ignore an explicitly specified default component value in an aggregate.

The intent was that the rule of 3.3.1(13/3) would be used when Default_Component_Value is specified, but that doesn't happen because the wording of 4.3.3(23.1/2) only talks about the component subtype (and not the subtype of the array as a whole).

Note that this rule is formally inconsistent with the published Ada 2012 standard (in that the initial value can be different than specified by that standard). This rule is better as it matches the understanding, the rule for record aggregates (where default expressions are used, of course), and it also matches the behavior of all known Ada 2012 implementations. (It still will be documented as an inconsistency, of course.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0139**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **4.4; 11.2; 11.3; 11.4.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0139 Raise expressions

Working Reference Number AI12-0022-1

Question

We want to encourage the conversion of comments into predicates/preconditions in existing libraries. However, changing the exception raised is likely to be an unacceptable incompatibility. How can this be done?

Summary of Response

The raise_expression is added to Ada.

Corrigendum Wording

Replace 4.4(3):

relation ::=
simple_expression [relational_operator simple_expression]
| simple_expression [**not**] **in** membership_choice_list

by:

relation ::=
simple_expression [relational_operator simple_expression]
| simple_expression [**not**] **in** membership_choice_list
| raise_expression

Insert before 11.2(6):

A choice with an *exception*_name *covers* the named exception. A choice with **others** covers all exceptions not named by previous choices of the same handled_sequence_of_statements. Two choices in different exception_handlers of the same handled_sequence_of_statements shall not cover the same exception.

the new paragraph:

An *exception*_name of an exception_choice shall denote an exception.

Insert after 11.3(2):

raise_statement ::= **raise**; |
raise *exception*_name [**with** *string*_expression];

the new paragraph:

raise_expression ::= **raise** *exception*_name [**with** *string*_expression]

Replace 11.3(3):

The name, if any, in a raise_statement shall denote an exception. A raise_statement with no *exception*_name (that is, a *re-raise statement*) shall be within a handler, but not within a body enclosed by that handler.

by:

An *exception*_name of a raise_statement or raise_expression shall denote an exception. A raise_statement with no *exception*_name (that is, a *re-raise statement*) shall be within a handler, but not within a body enclosed by that handler.

Replace 11.3(3.1):

The expression, if any, in a raise_statement, is expected to be of type String.

by:

A *string*_expression of a raise_statement or raise_expression is expected to be of type String.

A raise_expression is expected to be of any type.

Replace 11.3(4):

To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception*_name, the named exception is raised. If a *string*_expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

by:

To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception*_name, the named exception is raised. Similarly, for the evaluation of a raise_expression, the named exception is raised. In both of these cases, if a *string*_expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

Replace 11.4.1(10.1):

Exception_Message returns the message associated with the given Exception_Occurrence. For an occurrence raised by a call to Raise_Exception, the message is the Message parameter passed to Raise_Exception. For the occurrence raised by a raise_statement with an *exception*_name and a *string*_expression, the message is the *string*_expression. For the occurrence raised by a raise_statement with an *exception*_name but without a *string*_expression, the message is a string giving implementation-defined information about the exception occurrence. For an occurrence originally raised in some other manner (including by the failure of a language-defined check), the message is an unspecified string. In all cases, Exception_Message returns a string with lower bound 1.

by:

Exception_Message returns the message associated with the given Exception_Occurrence. For an occurrence raised by a call to Raise_Exception, the message is the Message parameter passed to Raise_Exception. For the occurrence raised by a raise_statement or raise_expression with an *exception*_name and a *string*_expression, the message is the *string*_expression. For the occurrence raised by a raise_statement or raise_expression with an *exception*_name but without a *string*_expression, the message is a string giving implementation-defined information about the exception occurrence. For an occurrence originally raised in some other manner (including by the failure of a language-defined check), the message is an unspecified string. In all cases, Exception_Message returns a string with lower bound 1.

Discussion

The intent is that the semantics of a raise_expression is the same as calling a function of the form:

```
function <raise_expression> returns <any type> is
begin
    raise exception_name [with string_expression];
    return X:<any type>; -- Junk return required by Ada 83 through at least 2012
end <raise_expression>;
```

A raise_expression has the precedence of a relation. This means that it will need to be parenthesized in most contexts. But it will not need to be parenthesized when used directly in a context like a conditional expression or return statement. It also will not need to be parenthesized when used directly with a logical operator/operation (and, and then, etc.).

For instance: (if Mode /= In_File then raise Mode_Error) is preferable to: (if Mode /= In_File then (raise Mode_Error))

We can't allow a raise_expression to go unparenthesized in all contexts, as there is an ambiguity with the optional string_expression. Does raise Some_Error with "aaa" & "bbb" mean (raise Some_Error with "aaa") & "bbb" or (raise Some_Error with "aaa" & "bbb")

We avoid this situation by using precedence so that the raise_expression has to be surrounded by parentheses if used with the "&" operator.

A raise_expression resolves to "any type". That means it might be necessary to qualify it in some circumstances, but the need for that should be rare.

It was suggested that the type be "any Boolean type", but that limits the usefulness of the construct in conditional expressions. For instance, imagine the following expression function:

```
function Foo (Bar : in Natural) return Natural is
    (case Bar is
        when 1 => 10,
        when 2 => 20,
        when others => (raise Program_Error)));
```

This is allowed as a raise_expression resolves to "any type"; if it resolved to "any Boolean type", some junk expression like "and True" would have to appended to make it legal -- which would do nothing for readability or understandability.

In addition, resolving to "any type" also solves the problem posed in AI12-0029-1, as that means "return raise Not_Implemented_Error;" is legal for any function. This makes it an easy idiom to use for functions that (temporarily) always raise an exception.

Example: Imagine the following routine in a GUI library:

```
procedure Show_Window (Window : in out Root_Window);
   -- Shows the window.
   -- Raises Not_Valid_Error if Window is not valid.
```

We would like to be able to use a predicate to check the comment. With the "raise_expression" we can do this without changing the semantics:

```
subtype Valid_Root_Window is Root_Window
   with Dynamic_Predicate =>
       Is_Valid (Valid_Root_Window) or else raise Not_Valid_Error;

procedure Show_Window (Window : in out Valid_Root_Window);
   -- Shows the window.
```

If we didn't include the raise_expression here, using the predicate would change the exception raised on this failure. That could cause the exception to fall into a different handler than currently, which is likely to not be acceptable.

An alternative way to write the predicate might be preferable:

```
subtype Valid_Root_Window is Root_Window
   with Dynamic_Predicate =>
       (if not Is_Valid (Valid_Root_Window) then raise Not_Valid_Error);
```

Similarly, the various Containers packages in Ada could use predicates or preconditions in this way to make some of the needed checks; but that can only be done if the semantics remains unchanged (raising Program_Error and Constraint_Error, not Assertion_Error). (The !proposal also shows how this could be used in Text_IO and other I/O packages.)

We considered a number of other alternatives to fix this problem:

Alternative #1: There is an optional "exception" clause on predicates and preconditions. This specifies the exception that will be raised on the failure of the check.

Alternative #2: There is an aspect "Pre_Exception" that specifies the exception to raise for Pre, and similarly for other assertions.

Alternative #3: Do nothing. The user can write a function that works like the proposed raise expressions: function Raise_Mode_Error (For_File : File_Type) return Boolean is begin raise Mode_Error with Name (For_File); return False; -- At least one return is required (see AI12-0029-1). end Raise_Mode_Error;

The problem with both of the first two alternatives is that the interface of a subprogram may include multiple exceptions that need to be checked:

```
procedure Put (File : in File_Type; Str : in String)
   with Pre => (Is_Open(File) or else raise Status_Error) and then
               (Mode(File) = Out_File or else
                raise Mode_Error with "Cannot read " & Name(File));
```

This cannot be handled with a single exception clause or aspect. This particular problem could be addressed by making the Status_Error check into a predicate, but that isn't likely to always work.

Another way to address the problem using an exception clause or aspect would be to allow multiple Pre aspects on a single declaration:

```
procedure Put (File : in File_Type; Str : in String)
   with Pre => Is_Open(File), Pre_Exception => Status_Error,
        Pre => Mode(File) = Out_File, Pre_Exception => Mode_Error;
```

but notice that the order of declaration of the preconditions is significant here, which is likely to be confusing (and a significant change from the current rules, where the order of evaluation of preconditions is unspecified).

In addition, we've lost the exception message that gives the name of the file with the wrong mode. That's a loss; we could introduce another aspect to deal with that, but by now it's clear that this solution is not "simpler" by any stretch of imagination.

The problem with the last alternative is the need to clutter the program with a number of exception raising functions. This is especially problematic in existing packages (like the language-defined ones) where adding these routines may not be allowed.

Thus the selected alternative seems clearly to be the best option.

This AI is classified as a Binding Interpretation in order that it can be implemented in and considered part of Ada 2012. We believe that writing preconditions and predicates without this facility would be a significant limitation when applied to existing packages (whether language-defined, implementation-defined, 3rd-party-defined, or user-defined) -- as changing the exception raised for some error changes the interface of a routine.

We do not intend this to set a precedent in the sense that additions (especially additions to syntax) should generally be considered only for future versions of Ada. We believe this is different in that it was considered an important facility for Ada 2012; it was removed from AI05-0290-1 only because the solutions considered didn't have sufficient maturity to include. Early practice using Ada 2012 has only reinforced the importance of this feature in writing preconditions and predicates; now that the feature is mature, we want it available in Ada 2012 compilers.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0140**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0140 Incompatibility of hidden untagged record equality

Working Reference Number AI12-0101-1

Question

Consider:

```
package P is
    type Priv is private;
private
    type Priv is record ...
    function "=" (L, R : Priv) return Boolean;
end P;
```

The Ada 2012 rule 4.5.2(9.8/3) says that this is illegal. This sort of construct appears fairly often in real code.

Do we want this incompatibility? (No.)

Summary of Response

Delete the legality rule about hidden untagged equality.

Corrigendum Wording

Replace 4.5.2(9.8):

> If the profile of an explicitly declared primitive equality operator of an untagged record type is type conformant with that of the corresponding predefined equality operator, the declaration shall occur before the type is frozen. In addition, if the untagged record type has a nonlimited partial view, then the declaration shall occur in the visible part of the enclosing package. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit.

by:

> If the profile of an explicitly declared primitive equality operator of an untagged record type is type conformant with that of the corresponding predefined equality operator, the declaration shall occur before the type is frozen. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit.

Discussion

The reason that 4.5.2(9.8/3) was originally added was for the freezing case. We cannot allow declaration of primitive equality for an untagged record type in a package body, since users of the type as a composite component could not know about the redefined equality that they must use to construct "=". Any call on "=" for the type or any type that has it as a component will freeze the type, so that is an appropriate point to stop further redeclarations of "=".

OTOH, the part about not allowing hidden "=" declarations for private types is purely about reducing the amount of inconsistency in the runtime behavior of "=". For Ada 2012, the meaning of "=" for a record type R cannot depend on visibility. If it did, then the effect of "=" for composite types with components of R would have to change based on visibility of the full type. Alternatively, we could make the composition not depend on visibility and leave the existing rule for direct use of "=", but that would open the door for "=" on stand-alone R and "=" on R as the only component of a record to give different results.

If we simply delete the second sentence of 4.5.2(9.8/3) (as proposed here), then 4.5.2(15/3) comes into effect and any overriding of "=" [before freezing] is used as the definition of "=" everywhere (even where the overriding "=" is not visible). This makes Ada 2012 inconsistent with previous versions of Ada.

However, the legality rule makes programs illegal even if they never call "=" on the private type (they might call "=" on the full type, which calls the overridden "=" in all versions of Ada, including Ada 2012). That

means that programs which will exhibit no inconsistency will be rejected unnecessarily. Moreover, even if there is an inconsistency, it's likely to fix bugs (the original justification for AI05-0123-1). It's hard to imagine why someone would want to (re)define "=" (presumably getting a different answer than the original predefined "=") and then have it ignored by clients. If they're doing that intentionally, it is very tricky code better handled some other way.

Finally, this inconsistency is (ahem) consistent with the inconsistencies introduced by AI05-0123-1. One expects users to think that all record equality works the same way; there's no value to have one case that still works differently (especially as it makes untagged records somewhat less capable than their tagged counterparts).

For all of these reasons, we decided to accept a bit more inconsistency rather than stomaching an incompatibility which may not be a problem at all.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0141**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **4.5.8**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0141 Definition of quantified expressions

Working Reference Number AI12-0158-1

Question

What is the value of a quantified expression if the array has zero elements? Is it always True? (Yes, for All.) Always False? (Yes, for Some.)

The RM says: "if the quantifier is all, the expression is True if the evaluation of the predicate yields True for each value of the loop parameter. It is False otherwise." It is definitely not clear to me, but this wording seems to lean toward being treated as False.

Summary of Response

Reword the semantics of quantified expressions to reduce confusion.

Corrigendum Wording

Insert before 4.5.8(1):

quantified_expression ::= **for** quantifier loop_parameter_specification => predicate
| **for** quantifier iterator_specification => predicate

the new paragraph:

Quantified expressions provide a way to write universally and existentially quantified predicates over containers and arrays.

Replace 4.5.8(6):

For the evaluation of a quantified_expression, the loop_parameter_specification or iterator_specification is first elaborated. The evaluation of a quantified_expression then evaluates the predicate for each value of the loop parameter. These values are examined in the order specified by the loop_parameter_specification (see 5.5) or iterator_specification (see 5.5.2).

by:

For the evaluation of a quantified_expression, the loop_parameter_specification or iterator_specification is first elaborated. The evaluation of a quantified_expression then evaluates the predicate for the values of the loop parameter in the order specified by the loop_parameter_specification (see 5.5) or iterator_specification (see 5.5.2).

Replace 4.5.8(8):

- If the quantifier is **all**, the expression is True if the evaluation of the predicate yields True for each value of the loop parameter. It is False otherwise. Evaluation of the quantified_expression stops when all values of the domain have been examined, or when the predicate yields False for a given value. Any exception raised by evaluation of the predicate is propagated.

by:

- If the quantifier is **all**, the expression is False if the evaluation of any predicate yields False; evaluation of the quantified_expression stops at that point. Otherwise (every predicate has been evaluated and yielded True), the expression is True. Any exception raised by evaluation of the predicate is propagated.

Replace 4.5.8(9):

- If the quantifier is **some**, the expression is True if the evaluation of the predicate yields True for some value of the loop parameter. It is False otherwise. Evaluation of the quantified_expression stops when all values of the domain have been examined, or when the predicate yields True for a given value. Any exception raised by evaluation of the predicate is propagated.

by:

- If the quantifier is **some**, the expression is True if the evaluation of any predicate yields True; evaluation of the quantified_expression stops at that point. Otherwise (every predicate has been evaluated and yielded False), the expression is False. Any exception raised by evaluation of the predicate is propagated.

Discussion

AARM 4.5.8(8.a/3) gives the intended answer (True) for quantifier all. How this supposed ramification is derived is not crystal clear.

The ramifications follow from the standard mathematical meaning of the forall and exists set operators. However, that may not be obvious to readers with less formal mathematics knowledge. We don't want Ada to only be understandable to a handful of elite people.

An additional issue was raised during the discussion of wording alternatives for this bullet: the wording makes it sound like all of the predicates are evaluated before the answer is determined. It then follows with a sentence that effectively says to forget that, actually it is short-circuited. It would be better to tell the truth in the first place.

Thus, we adopt wording which says what the operations do without any attempt to make them appear to be similar to some set operation. That sort of thing is best done in the introduction, rather than trying to bend the dynamic semantics to serve that purpose. We add an introductory paragraph to serve this purpose.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0142**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.6; 6.4.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0142 View conversions and out parameters passed by copy

Working Reference Number AI12-0074-1

Question

6.4.1(12-15) reads in part:

> For an out parameter that is passed by copy, the formal parameter object is created, and: ... For a scalar type that has the Default_Value aspect specified, the formal parameter is initialized from the value of the actual, without checking that the value satisfies any constraint or any predicate

This presupposes that "the value of the actual" exists and is well defined.

How does this work in the case where the actual parameter is a view conversion? Consider:

```
with Text_IO; use Text_IO;
procedure Bad_Conv1 is
  type Defaulted_Integer is new Integer
    with Default_Value => 123;

  procedure Foo (x : out Integer) is begin null; end;

  Y : Long_Long_Float := Long_Long_Float'Last;
   -- or, for a variation, leave Y uninitialized
begin
 Foo (Defaulted_Integer (Y));  -- Now illegal.
 Put_Line (Long_Long_Float'Image (Y));
end;

with Text_IO; use Text_IO;
procedure Bad_Conv2 is

  type Mod3 is mod 3 with Default_Value => 0;
  type Mod5 is mod 5;

  procedure Foo (xx : out Mod3) is begin null; end;

  YY : Mod5 := 4;
begin
 Foo (Mod3 (YY));  -- Now illegal.
 Put_Line (Mod5'Image (YY));
end;
```

In this case, the converted value might not (probably won't) fit into the parameter representation. What should happen? (The calls on Foo are illegal.)

Summary of Response

Scalar view conversion values are well defined if the Default_Value aspect of the target type is specified. Access type view conversions are not allowed for unrelated types.

Corrigendum Wording

Replace 4.6(56):

- Reading the value of the view yields the result of converting the value of the operand object to the target subtype (which might raise Constraint_Error), except if the object is of an access type and the view conversion is passed as an **out** parameter; in this latter case, the value of the operand object is used to initialize the formal parameter without checking against any constraint of the target subtype (see 6.4.1).

by:

- Reading the value of the view yields the result of converting the value of the operand object to the target subtype (which might raise Constraint_Error), except if the object is of an elementary type and the view conversion is passed as an **out** parameter; in this latter case, the value of the operand object may be used to initialize the formal parameter without checking against any constraint of the target subtype (as described more precisely in 6.4.1).

Insert after 6.4.1(5):

If the mode is **in out** or **out**, the actual shall be a name that denotes a variable.

the new paragraph:

If the mode is **out**, the actual parameter is a view conversion, and the type of the formal parameter is an access type or a scalar type that has the Default_Value aspect specified, then

- there shall exist a type (other than a root numeric type) that is an ancestor of both the target type and the operand type; and
- in the case of a scalar type, the type of the operand of the conversion shall have the Default_Value aspect specified.

In addition to the places where Legality Rules normally apply (see 12.3), these rules also apply in the private part of an instance of a generic unit.

Replace 6.4.1(13.1):

- For a scalar type that has the Default_Value aspect specified, the formal parameter is initialized from the value of the actual, without checking that the value satisfies any constraint or any predicate;

by:

- For a scalar type that has the Default_Value aspect specified, the formal parameter is initialized from the value of the actual, without checking that the value satisfies any constraint or any predicate, except in the following case: if the actual parameter is a view conversion and either
  - there exists no type (other than a root numeric type) that is an ancestor of both the target type and the type of the operand of the conversion; or
  - the Default_Value aspect is unspecified for the type of the operand of the conversion

  then Program_Error is raised;

Response

(See !summary.)

Discussion

The value of an elementary view conversion is that of its operand.

We want the value of an elementary view conversion to be a well defined value of the target type of the conversion if the target type is either an access type or a scalar type for whose Default_Value aspect has been specified.

4.6(56) is supposed to define this, but it isn't close to correct. First, it was never updated to include predicates and null exclusions as 6.4.1(13/3) was. Second, it doesn't include the rules for scalar types with Default_Values specified at all. Third, it doesn't explain what happens if the original value does not fit in the representation of the target type (as in the example in the question).

Note that the "does not fit" case already exists in Ada, all the way back to Ada 95. Specifically, if an implementation has multiple representations for access types, and the view conversion is going from the larger representation to the smaller one, then the problem could occur. [This happened historically; the segmented architecture of the 16-bit 8086 led to compilers supporting both long (with segments) and short (without segments) pointers. Similarly, the Unisys U2200 compiler that the author worked on supported Ada pointers

(machine addresses [which accessed words]) and C pointers (byte addresses, a pair of a machine address and a byte offset).]

Clearly, the "does not fit" problem was unusual for access types, so it's not surprising that it was never solved. However, it becomes much more important for scalar types with Default_Values, especially when the source object is of a type that does not have a Default_Value (and thus might be uninitialized).

To ensure that the value of a scalar view conversion is well-defined, we disallow such scalar view conversions in cases where either - the set of values of the type of the operand and of the target type need not be the same; or - an object of the type of the operand might not have a well defined value.

The rule that was chosen to accomplish this is that the two types must have a common ancestor type and, in the scalar case, the operand type's Default_Value aspect must be specified.

This is expressed both as a Legality Rule (which handles most cases statically) and as a runtime check (which handles certain obscure cases involving generic instance bodies which make it past the legality rule). For an implementation which macro-expands instance bodies, the outcome of this runtime check is always known statically.

The runtime check would be triggered in examples like the following:

```
declare
   type T is range 0 .. 100;
   type T_With_Default is new T with Default_Value => 0;

   generic
      type New_T is new T;
   package G is
   end G;

   package body G is
      X : T;

      procedure P (X : out New_T) is
      begin
         X := 0;
      end;
   begin
      P (New_T (X)); -- conversion raises Program_Error
   end G;

    package I is new G (T_With_Default);

 begin
    null;
 end;
```

This Legality Rule is compatible with previous versions of Ada as they do not have the Default_Value aspect. No calls on inherited subprograms will be affected, as 13.1(10) ensures that the Default_Value aspect is the same for the derived type and the parent type, and of course the types are related in this case. That means that all affected view conversions will be explicit in the program.

We use the same Legality Rule to handle the more minor issue with access types. Note that this will be incompatible in rare cases (access type view conversions are rare in existing code, and ones that would trigger the rule should be much rarer). Such code was not possible in Ada 83 (unrelated access type conversions didn't exist); Ada 95 general access conversions are required to trigger the problem. Also note that unlike scalar types, it is not that uncommon to read an "out" access parameter, in order to check bounds or discriminants of the designated object. This means that the problematic case is more likely to occur in practice.

An alternative solution was to initialize the parameter with the Default_Value in this case. However, that would potentially "deinitialize" the actual object in the view conversion case:

```
type Small is range 0 .. 255
```

```
   with Default_Value => 0;

procedure P (Do_It : in Boolean; Result : out Small) is
begin
    if Do_It then
        Result := 10;
    end if;
end P;

Obj : Positive := 1;

P (Do_It => False, Result => Small(Obj)); -- (1)
```

The call to P at (1) would raise Constraint_Error upon return, even though the previously initialized value of Obj is properly initialized and it wasn't changed. This does not seem acceptable, especially as the similar rules for access types do not have this effect.

A different alternative solution would be to declare the parameter to be abnormal in cases where the legality rule would be triggered. This would make the program erroneous if (and only if) the parameter was actually read (which doesn't happen for many out parameters). The problem with this is that it introduces a new kind of erroneous execution.

A possibility to minimize both the incompatibility and the possibility of erroneousness is to use a suppressible error here (presuming that we define those, see AI12-0092-1). In that case, the programmer would have to take explicit action (suppress the error) to cause the possibility of erroneous execution. That might be a better solution for this case (because the incompatibility can be easily avoided by suppressing the error as well as by introducing a temporary), but it depends on us having decided to define suppressible errors. We ultimately determined that this case is rather unlikely to occur, so that it doesn't provide a reason to define suppressible errors.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0143**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0143 The exception raised when a subtype conversion fails a predicate check

Working Reference Number AI12-0096-1

Question

4.6(57/3) says:

If an Accessibility_Check fails, Program_Error is raised. If a predicate check fails, Assertions.Assertion_Error is raised. Any other check associated with a conversion raises Constraint_Error if it fails.

However, AI12-0054-2 defines the Predicate_Failure aspect, which can change what exception is raised when a predicate check fails. Do we need to adjust this wording? (Yes.)

Summary of Response

A subtype conversion that fails a predicate check has the effect defined in 3.2.4; it does not unconditionally raise Assertion_Error.

Corrigendum Wording

Replace 4.6(57):

> If an Accessibility_Check fails, Program_Error is raised. If a predicate check fails, Assertions.Assertion_Error is raised. Any other check associated with a conversion raises Constraint_Error if it fails.

by:

> If an Accessibility_Check fails, Program_Error is raised. If a predicate check fails, the effect is as defined in subclause 3.2.4, "Subtype Predicates". Any other check associated with a conversion raises Constraint_Error if it fails.

Discussion

Surely we don't want one set of rules for the effect of a predicate check in 3.2.4 and a different set in 4.6. The fact that this is the case is clearly an oversight (which has occurred repeatedly with this paragraph).

We don't want to repeat the relatively complex rules for the evaluation of the Predicate_Failure aspect anywhere, so we simply refer to them here. (At least that will eliminate one potential problem in the future.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0144**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.7**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0144 A qualified expression makes a predicate check

Working Reference Number AI12-0100-1

Question

If the subtype of a qualified expression has a predicate, is that predicate checked? (Yes.)

4.7(4) says that the evaluation of a qualified_expression "...checks that its value belongs to the subtype denoted by the subtype_mark." The AARM note 4.7(4.a) makes it clear that a subtype conversion is NOT performed here, so the predicate check associated with a subtype conversion doesn't happen here.

3.2(8) defines "belongs" for this context: "The set of values of a subtype consists of the values of its type that satisfy its constraint and any exclusion of the null value. Such values belong to the subtype."

As we know, a predicate doesn't change the set of values. So it doesn't appear here, either.

Thus there is no predicate check for a qualified expression. This seems weird, since a similar type conversion would make a predicate check. Should the wording make it clear that a qualified expression does make a predicate check? (Yes.)

Summary of Response

A qualified expression checks that the value satisfies any predicates of the qualified subtype.

Corrigendum Wording

Replace 4.7(4):

> The evaluation of a qualified_expression evaluates the operand (and if of a universal type, converts it to the type determined by the subtype_mark) and checks that its value belongs to the subtype denoted by the subtype_mark. The exception Constraint_Error is raised if this check fails.

by:

> The evaluation of a qualified_expression evaluates the operand (and if of a universal type, converts it to the type determined by the subtype_mark) and checks that its value belongs to the subtype denoted by the subtype_mark. The exception Constraint_Error is raised if this check fails. Furthermore, if predicate checks are enabled for the subtype denoted by the subtype_mark, a check is performed as defined in subclause 3.2.4, "Subtype Predicates" that the value satifies the predicates of the subtype.

Discussion

We certainly want the checks made by a qualified expression to be at least as strict as those associated with an explicit type conversion (the type conversion will allow additional things where a conversion happens).

For example:

```
subtype Even is Natural with Dynamic_Predicate => Even mod 2 = 0;

Var   : Natural;
Three : constant Natural := 3;

Var := Even'(Three); -- Needs new wording for a predicate check.

Var := Even(Three); -- Predicate check, thus raises Assertion_Error.
```

Without the new wording, Even'(Three) would not raise an exception, while Even(Three) would. [Aside: Even'(3) would raise an exception, as in that case 3 would have a universal type and that would be subtype converted, thus checking the predicate.]

We need to refer to the effect defined in 3.2.4 for the same reason that we do that in 4.6 (see AI12-0096-1). We don't want to repeat the somewhat complex rules as to what happens when a predicate check fails.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0145**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **5.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0145 Meaning of subtype_indication in array component iterators

Working Reference Number AI12-0151-1

Question

Consider an array component iterator with a specified subtype. 5.5.2(5/3) says that the specified subtype has to cover the component type of the array object. 5.5.2(7/3) says that the nominal subtype of the loop parameter is that specified. So consider the following:

```
type Arr is array (1 .. 10) of Integer;

subtype Short is Integer range 1 .. 10;

Obj : Arr := (2,4,6,8,10,12,14,16,18,20);

for P : Short of Obj loop
    case P is
       when 1 .. 9 => ...
       when 10 => ...
    end case; -- Legal, no others needed or allowed.
end loop;
```

The nominal subtype of P is Short, so the case statement does not allow any others clause. However, some of the elements have values outside of this nominal subtype. Essentially, the language is requiring that the loop parameter has invalid values, as there is no check to prevent this.

What is the intent here? (The subtype must statically match.)

Summary of Response

The optional subtype specified in an array component iterator or a container element iterator must statically match the array component subtype or container element subtype.

Corrigendum Wording

Replace 5.5.2(5):

> The type of the subtype_indication, if any, of an array component iterator shall cover the component type of the type of the *iterable*_name. The type of the subtype_indication, if any, of a container element iterator shall cover the default element type for the type of the *iterable*_name.

by:

> The subtype defined by the subtype_indication, if any, of an array component iterator shall statically match the component subtype of the type of the *iterable*_name. The subtype defined by the subtype_indication, if any, of a container element iterator shall statically match the default element subtype for the type of the *iterable*_name.

Discussion

We have essentially three choices:

(A) The subtype of the loop parameter works like that of an object renames; it is essentially ignored and the component subtype used instead.

(B) The loop parameter is a view conversion of the component, meaning that there are extra checks when it is created and assigned.

(C) The subtype of the loop parameter has to statically match that of the component subtype.

(A) is misleading to the reader; the subtype doesn't really mean what it says. [This is a long-time complaint about renames; compatibility concerns have prevented us from fixing them, but the consensus is that they're a bad design.]

(B) means that one might get exceptions raised by the loop itself if the wrong subtype is written by accident. Moreover, there has to be some sort of reverse check when the component is assigned. In addition, such a check would be problematical of the components are uninitialized. A loop to initialize an array such as: for Comp : Natural of Obj loop Comp := 10; end loop; might raise an exception if the uninitialized junk has a negative value.

(C) is incompatible with the Ada 2012 language as defined. The wording in the Ada 2012 standard clearly allows constructions such as in the example, but requiring static matching would render it illegal.

Given the potential for confusion, and the early state of Ada 2012 adoption, we chose to require static matching. The alternatives either are full of potential landmines for the language definition and the user (B), or are misleading to the reader (A) and not really helpful (since any constraints are ignored, why write them in the first place?).

Container element iterators have similar issues with the subtype name as array component iterators. We adopt the same solution as for arrays for them.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0146**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **5.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0146 Generalized iterators and discriminant-dependent components

Working Reference Number AI12-0047-1

Question

1) Consider:

```
subtype Index is Integer range 0 .. 100;

type Rec (Last : Index := 50) is
   record F : String (1 .. Last); end record;

X : Rec;
begin
for Char of X.F loop
   X := (0, "");
   Char := '?';
end loop;
```

In this case, the assignment to X makes the array object disappear. This should be illegal, right? (Yes.)

2) The following is defined to be a master (7.6.1(3/2)):

> ... the evaluation of an expression, function_call, or range that is not part of an enclosing expression, function_call, range, or simple_statement other than a simple_return_statement

Consider:

> for I of Func_Returning_Array_With_Controlled_Elements loop

The rule above means that the function call result object gets finalized before beginning the first iteration of the loop. This is not good (especially as the tampering mechanism assumes that this finalization happens when the loop is exited).

Should something be changed? (No.)

Summary of Response

The iterable_name or iterator_name of an iterator cannot be a discriminant-dependent subcomponent of a mutable type.

The iterable_name or iterator_name of an iterator is not finalized until the loop completes.

Corrigendum Wording

Insert after 5.5.2(6):

> In a container element iterator whose *iterable*_name has type *T*, if the *iterable*_name denotes a constant or the Variable_Indexing aspect is not specified for *T*, then the Constant_Indexing aspect shall be specified for *T*.

the new paragraph:

> The *iterator*_name or *iterable*_name of an iterator_specification shall not denote a subcomponent that depends on discriminants of an object whose nominal subtype is unconstrained, unless the object is known to be constrained.

Discussion

Problem #1 is solved by following the lead of renaming. We directly repeat the interesting rule as defining iterators in terms of renaming would drag in other rules that may or may not be appropriate. In particular, it would require that the object can be clobbered inside the loop (with unpredictable results for a container iterator):

for E of X loop ... X := <something>; ... end loop;

Ada.Containers has tampering checks to prevent this situation (the assignment would raise Program_Error), but user-defined packages may not. So we do not take this approach.

Problem #2 is not a problem at all, as the master of a return object of a function call is that which encloses the function call; whether the function call is a master has no impact on that. (See AARM 7.6.1(3.c/2).) The fact that the function call is a master mainly is important to specify the lifetime of the actual parameters to be short, as in the following example:

for E of Func_Returning_Array (Func_Returning_Record_With_Controlled_Components) loop

In this case, the result of Func_Returning_Record_With_Controlled_Components is finalized before the loop begins to execute (assuming the parameter to Func_Returning_Array is not aliased), as the call of Func_Returning_Array is its master, while the result of Func_Returning_Array is not finalized until the loop is complete (as the loop statement is the master). This is what we want for generalized iterators, so no change is needed.

There is no problem with more complex names, either. Consider: for E of Func.Some_Array_with_Controlled_Components loop An iterator_specification contains an iterator_name or iterable_name; these are not expressions. As such, the entire name is not a master, either (a name is not in the list of the things that can be masters in 7.6.1(3/2)). Thus, in this case, Func.Some_Array_with_Controlled_Components is not a master, so its master is the entire loop statement; thus the master of the result of Func is also the entire loop statement, and the result of Func is not finalized until the iteration completes. This is what we want.

Contrast this with the following: for I in 1 .. Func.Some_Integer loop Here, Func.Some_Integer is an expression, this is a master by 7.6.1(3/2) as it is not enclosed by one of the things in the list in that rule. Thus the master of the result of Func is the master that is the Func.Some_Integer, and thus that result is finalized before the loop iterates. Again, this is what we want.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0147**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **5.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0147 Legality and exceptions of generalized loop iteration

Working Reference Number AI12-0120-1

Question

The original version of a proposed ACATS test used a default iterator function with the specification:

```
function Iterate
  (Container : aliased in out Sparse_Array)
   return Sparse_Array_Iterator_Interfaces.Reversible_Iterator'Class;
```

This caused an internal error in a compiler when the test tried iterating on a function call (that is, a constant):

```
for Item of Get_Constant_Sparse_Data loop
```

We eventually realized that this expands to

```
for Cur in Get_Constant_Sparse_Data.Iterate loop
```

which is of course illegal (as we're passing a function call to an in out parameter).

There is no rule which disallows this, but clearly we don't want to allow making calls that would otherwise be illegal. Do we need a new rule? (Yes.)

Summary of Response

If the call to the default iterator function would be illegal, a container element iterator is illegal. If the cursor type for a generalized iterator or container element iterator is limited, the iterator is illegal. If any of the calls to functions in a generalized iterator or container element iterator propagate an exception, the enclosing loop statement propagates the exception.

Corrigendum Wording

Insert after 5.5.2(6):

In a container element iterator whose *iterable*_name has type *T*, if the *iterable*_name denotes a constant or the Variable_Indexing aspect is not specified for *T*, then the Constant_Indexing aspect shall be specified for *T*.

the new paragraphs:

A container element iterator is illegal if the call of the default iterator function that creates the loop iterator (see below) is illegal.

A generalized iterator is illegal if the iteration cursor subtype of the *iterator*_name is a limited type at the point of the generalized iterator. A container element iterator is illegal if the default cursor subtype of the type of the *iterable*_name is a limited type at the point of the container element iterator.

Insert after 5.5.2(13):

For a forward container element iterator, the operation First of the iterator type is called on the loop iterator, to produce the initial value for the loop cursor. If the result of calling Has_Element on the initial value is False, then the execution of the loop_statement is complete. Otherwise, the sequence_of_statements is executed with the loop parameter denoting an indexing (see 4.1.6) into the iterable container object for the loop, with the only parameter to the indexing being the current value of the loop cursor; then the Next operation of the iterator type is called with the loop iterator and the loop cursor to produce the next value to be assigned to the loop cursor. This repeats until the result of calling Has_Element on the loop cursor is False, or until the loop is left as a consequence of a transfer of control. For a reverse container element iterator, the operations Last and Previous are called rather than First and Next. If the loop parameter is a constant (see above), then the indexing uses the default constant indexing function for the type of the iterable container object for the loop; otherwise it uses the default variable indexing function.

the new paragraph:

> Any exception propagated by the execution of a generalized iterator or container element iterator is propagated by the immediately enclosing loop statement.

Discussion

We want this to work just like a macro-expansion into the appropriate code, so we define that it works that way.

The rules for the interfaces defined in Iterator_Interfaces prevents any of the calls to those functions from being illegal, so we don't need any rules for them.

We specify that an exception raised by one of the calls that are used to implement an iterator, or by an assignment used to implement an iterator, is propagated by the loop statement. This means that those exceptions cannot be handled inside of the loop; in particular, they cannot be handled by the sequence_of_statements.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0148**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **6.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0148 Pre- and Postconditions are allowed on generic subprograms

Working Reference Number AI12-0045-1

Question

6.1.1(1/3) says: For a subprogram or entry, the following language-defined aspects may be specified with an aspect_specification

and AARM 6.1(20.a/3) adds ... a generic subprogram is not a subprogram.

This implies that pre/post-condition aspect specifications are not allowed for generic subprograms. Should they be? (Yes.)

Summary of Response

Aspects Pre and Post can be specified on generic subprograms, but not on instances of generic subprograms.

Corrigendum Wording

Replace 6.1.1(1):

For a subprogram or entry, the following language-defined aspects may be specified with an aspect_specification (see 13.1.1):

by:

For a noninstance subprogram, a generic subprogram, or an entry, the following language-defined aspects may be specified with an aspect_specification (see 13.1.1):

Discussion

If we allowed Pre and Post on instances that are subprograms, we would be introducing a maintenance problem, as converting the generic subprogram to a generic package containing that subprogram would require the removal of the aspects (there is no corresponding feature for subprograms declared inside of package instances). The examples of such usage have not been compelling (they usually are refinements of overly general aspects on the generic).

In addition, if we allowed these aspects on both the generic (which is important) and on the instance, we'd have to define how these combine. That is a problem for preconditions, which should combine with "and" or "or" depending on who you are talking to and about what. ["or" is required to follow LSP.] To avoid answering this question, we make the programmer answer it explicitly. That principle should be continued.

We considered allowing the aspects on an instance if none are given on the generic unit in order to avoid this latter problem. But we decided against this because of the maintenance problem mentioned above.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0149**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **6.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0149 Class-wide preconditions and statically bound calls

Working Reference Number AI12-0113-1

Question

6.1.1(7.3) says that, a parameter having type T is interpreted as having type T'Class. That interpretation means that any call to a primitive operation of T is a dispatching call. Effectively, the parameter has an implicit conversion to T'Class.

For a statically bound call when the tag of the parameter identifies a different type than its nominal type, this means that a different body would be executed than would be executed for the equivalent Pre. That means that Pre'Class is not quite equivalent to a set of identical Pre aspects with an additional guarentee for future overridden subprograms. But that was the intent: that Pre'Class be LSP-safe but otherwise be equivalent to putting the same Pre on every related subprogram.

The implicit redispatching implied by 6.1.1(7/3) makes static analysis harder, as a tool cannot plug in the body of a subprogram called from Pre'Class even if the tool knows the contents of that body, because the call might dispatch to some other body.

Should this be corrected somehow? (Yes.)

Summary of Response

Class-wide preconditions, postconditions, and type invariants are processed using the actual types for the parameters of type T (not always using T'Class). In particular, for a statically bound call, the calls within the Pre'Class expression are statically bound.

Corrigendum Wording

Replace 6.1.1(7):

> Within the expression for a Pre'Class or Post'Class aspect for a primitive subprogram of a tagged type *T*, a name that denotes a formal parameter of type *T* is interpreted as having type *T*'Class. Similarly, a name that denotes a formal access parameter of type access-to-*T* is interpreted as having type access-to-*T*'Class. This ensures that the expression is well-defined for a primitive subprogram of a type descended from *T*.

by:

> Within the expression for a Pre'Class or Post'Class aspect for a primitive subprogram *S* of a tagged type *T*, a name that denotes a formal parameter (or *S*'Result) of type *T* is interpreted as though it had a (notional) type *NT* that is a formal derived type whose ancestor type is *T*, with directly visible primitive operations. Similarly, a name that denotes a formal access parameter (or *S*'Result) of type access-to-*T* is interpreted as having type access-to-*NT*. The result of this interpretation is that the only operations that can be applied to such names are those defined for such a formal derived type. This ensures that the expression is well-defined for any primitive subprogram of a type descended from *T*.

Replace 6.1.1(18):

> If a Pre'Class or Post'Class aspect is specified for a primitive subprogram of a tagged type *T*, then the associated expression also applies to the corresponding primitive subprogram of each descendant of *T*.

by:

> If a Pre'Class or Post'Class aspect is specified for a primitive subprogram *S* of a tagged type *T*, or such an aspect defaults to True, then a corresponding expression also applies to the corresponding primitive subprogram *S* of each descendant of *T*. The *corresponding expression* is constructed from the associated expression as follows:
>
> - References to formal parameters of *S* (or to *S* itself) are replaced with references to the corresponding formal parameters of the corresponding inherited or overriding subprogram *S* (or to the corresponding subprogram *S* itself).

The primitive subprogram *S* is illegal if it is not abstract and the corresponding expression for a Pre'Class or Post'Class aspect would be illegal.

Replace 6.1.1(37):

For any subprogram or entry call (including dispatching calls), the checks that are performed to verify specific precondition expressions and specific and class-wide postcondition expressions are determined by those for the subprogram or entry actually invoked. Note that the class-wide postcondition expressions verified by the postcondition check that is part of a call on a primitive subprogram of type *T* includes all class-wide postcondition expressions originating in any progenitor of *T*, even if the primitive subprogram called is inherited from a type *T1* and some of the postcondition expressions do not apply to the corresponding primitive subprogram of *T1*.

by:

For any subprogram or entry call *S* (including dispatching calls), the checks that are performed to verify specific precondition expressions and specific and class-wide postcondition expressions are determined by those for the subprogram or entry actually invoked. Note that the class-wide postcondition expressions verified by the postcondition check that is part of a call on a primitive subprogram of type *T* includes all class-wide postcondition expressions originating in any progenitor of *T*, even if the primitive subprogram called is inherited from a type *T1* and some of the postcondition expressions do not apply to the corresponding primitive subprogram of *T1*. Any operations within a class-wide postcondition expression that were resolved as primitive operations of the (notional) formal derived type *NT*, are in the evaluation of the postcondition bound to the corresponding operations of the type identified by the controlling tag of the call on *S*. This applies to both dispatching and non-dispatching calls on *S*.

Replace 6.1.1(38):

The class-wide precondition check for a call to a subprogram or entry consists solely of checking the class-wide precondition expressions that apply to the denoted callable entity (not necessarily the one that is invoked).

by:

The class-wide precondition check for a call to a subprogram or entry *S* consists solely of checking the class-wide precondition expressions that apply to the denoted callable entity (not necessarily to the one that is invoked). Any operations within such an expression that were resolved as primitive operations of the (notional) formal derived type *NT*, are in the evaluation of the precondition bound to the corresponding operations of the type identified by the controlling tag of the call on *S*. This applies to both dispatching and non-dispatching calls on *S*.

Discussion

Consider two nearly identical hierarchies of types. First:

```
package P11 is
    type Root is abstract tagged private;

    function Is_Valid (P : in Root) return Boolean;

    procedure Proc (P : in Root)
        with Pre => Is_Valid(P);

    procedure Proc2 (P : in Root);
private
    ...
end P11;

with P11;
package P12 is
    type Child is new P11.Root with private;

    overriding
    function Is_Valid (P : in Child) return Boolean;

    overriding
    procedure Proc (P : in Child)
        with Pre => Is_Valid(P);
```

```
    -- Proc2 inherited.
private
    ...
end P12;

with P12;
package P13 is
    type GrandChild is new P12.Child with private;

    overriding
    function Is_Valid (P : in Grandchild) return Boolean;

    overriding
    procedure Proc (P : in Grandchild)
        with Pre => Is_Valid(P);

    -- Proc2 inherited.
private
    ...
end P13;
```

and the second hierarchy is:

```
package P21 is
    type Root is abstract tagged private;

    function Is_Valid (P : in Root) return Boolean;

    procedure Proc (P : in Root)
        with Pre'Class => Is_Valid(P);

    procedure Proc2 (P : in Root);
private
    ...
end P21;

with P21;
package P22 is
    type Child is new P21.Root with private;

    overriding
    function Is_Valid (P : in Child) return Boolean;

    overriding
    procedure Proc (P : in Child);

    -- Proc2 inherited.
private
    ...
end P22;

with P22;
package P23 is
    type GrandChild is new P22.Child with private;

    overriding
    function Is_Valid (P : in Grandchild) return Boolean;

    overriding
    procedure Proc (P : in Grandchild);

    -- Proc2 inherited.
private
    ...
end P23;
```

The only difference between these hierarchies is that the first uses a specific Pre on each declaration of Proc, while the second uses a class-wide Pre on the first declaration of Proc and allows it to be inherited on the others.

Our intent is that these behave identically at runtime, with the only difference being that there is an additional guarantee that a new descendant of P21.Root will also have the Pre'Class expression (that has to be manually added in the other hierarchy).

The additional guarantee allows analyzing dispatching calls without knowing the exact body that they will reach, as we know that all bodies will have the precondition Pre'Class. Such analysis can only be done for the P11 hierarchy via full program analysis.

We define the meaning of Pre'Class in terms of a nominal formal derived type in order that only primitive operations of the type can be used in the expression. The original Ada 2012 definition of the types as T'Class also has a runtime effect that we do not want (as noted above).

In addition, defining the parameters as having T'Class introduces other problems. For instance, with a global variable: type T is tagged ...; G : T'Class := ...; function Is_Valid_2 (A, B : T) return Boolean; procedure Proc (X : T) with Pre'Class => Is_Valid_2 (X, G); If X is considered to have T'Class, this expression is legal (both operands being dynamically tagged). But this doesn't make sense for an inherited type: type T1 is new T with ...; -- inherits Is_Valid_2 (A, B : T1) return Boolean; -- inherits Proc (X : T) -- with Pre'Class => Is_Valid_2 (X, G); But this Pre'Class makes no sense, as G does not match type T1.

If we simply left the parameters having type T, then non-primitive operations of the type could be used in Pre'Class -- but descendant types have no such operations. We could have rechecked the Pre'Class for each later subprogram, but that causes problems if visibility has changed somehow (especially for class-wide operations).

On the other hand, a formal derived type has just the operations that we want to allow (primitive operations) but no others. In particular, G in the example above does not match the NT type, so the above problem can't happen.

We have to recheck the legality of the corresponding expression, just like in a generic instantiation. In particular, a call to a primitive subprogram could be illegal because it is abstract:

```
package Pkg3 is
   type T is abstract tagged null record;
   function Is_Ok (X : T) return Boolean is abstract;
   procedure Proc (X : T) with Pre'Class => Is_Ok (X); -- Illegal.
end Pkg3;
```

(Note that Proc could be called in a statically bound call with a type conversion of some object to T.)

We can't just check this once, because whether a routine is abstract can be changed when deriving:

```
package Pkg4 is
   type T is tagged null record;
   function Is_Ok (X : T) return Boolean;
   procedure Proc (X : T) with Pre'Class => Is_Ok (X); -- OK.
end Pkg4;

with Pkg4;
package Pkg5 is
   type NT is abstract new Pkg4.T null record;
   function Is_Ok (X : NT) return Boolean is abstract;
   -- inherits Proc, Pre'Class => Is_Ok (X); -- Illegal.
end Pkg5;
```

We don't check this for abstract routines, since a statically bound call is illegal, and there cannot be objects with the tag of an abstract type, so there can't be any dispatching calls that land there, either.

Note carefully that resolution is not redone (again like a generic instantiation); only the legality checks. The resolution of global variables and the like is unchanged from the original expression.

We've added a mention that even the implicit Pre'Class of True is inherited. That follows from the Liskov Substitution Principle -- the precondition of an overridden routine has to be the same or weaker than that of the parent routine. There is nothing weaker than True, so if a routine does not have an explicit Pre'Class, it can't usefully be given a Pre'Class later in the derivation tree. This topic is further explored in AI12-0131-1.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0150**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **6.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0150 Inherited Pre'Class when unspecified on initial subprogram

Working Reference Number AI12-0131-1

Question

Consider the following declarations:

```
type T is tagged ...
procedure P (A : in out T); -- (1)

type TT is new T with ...
function Is_OK (A : in TT) return Boolean;
procedure P (A : in out TT) with Pre'Class => Is_OK (A); -- (2)

S_Obj : TT;
T_Obj : T'Class := T'Class(S_Obj);

P (S_Obj); -- (3)
P (T_Obj); -- (4)
```

What precondition is evaluated for each of these calls? (Assume all are enabled and no exceptions are raised.)

For (3), the specified Pre'Class is evaluated; 6.1.1(18/3) says that no precondition is inherited (since none are specified on (1)). So the precondition that is evaluated is Is_OK(S_Obj).

For (4), there is no specified precondition (6.1.1(38/3) says that we don't care about the class-wide preconditions of the actual body, only the subprogram that is nominally invoked). However, we'll dispatch to the body of P, effectively giving us the same effect of the call (3) -- except that the precondition is different. The body of P cannot assume anything, as there is no effective precondition.

Note that the results would be different if (1) has explicitly been given a Pre'Class of True. Then the precondition of both calls would be effectively True. Something seems wrong here, should this be fixed? (Yes.)

Summary of Response

If the initial definition of a primitive subprogram of a tagged type does not specify Pre'Class, the corresponding subprograms of descendant types inherit a class-wide precondition of True.

Corrigendum Wording

Insert after 6.1.1(17):

> If a renaming of a subprogram or entry *S1* overrides an inherited subprogram *S2*, then the overriding is illegal unless each class-wide precondition expression that applies to *S1* fully conforms to some class-wide precondition expression that applies to *S2* and each class-wide precondition expression that applies to *S2* fully conforms to some class-wide precondition expression that applies to *S1*.

the new paragraphs:

> Pre'Class shall not be specified for an overriding primitive subprogram of a tagged type *T* unless the Pre'Class aspect is specified for the corresponding primitive subprogram of some ancestor of *T*.
>
> In addition to the places where Legality Rules normally apply (see 12.3), these rules also apply in the private part of an instance of a generic unit.

Replace 6.1.1(18):

> If a Pre'Class or Post'Class aspect is specified for a primitive subprogram of a tagged type *T*, then the associated expression also applies to the corresponding primitive subprogram of each descendant of *T*.

by:

If a Pre'Class or Post'Class aspect is specified for a primitive subprogram of a tagged type *T*, or such an aspect defaults to True, then the associated expression also applies to the corresponding primitive subprogram of each descendant of *T*.

Discussion

Note that specifying a class-wide precondition on (2) [the overriding subprogram] is strengthening the precondition, a violation of the Liskov Substitutability Principle [LSP]. But Pre'Class is intended to directly model LSP.

Thus we say that overriding a subprogram that has no Pre'Class specified for any ancestor causes the new subprogram to inherit a Pre'Class of True.

Note that this only applies to "root" subprograms that don't have Pre'Class; overriding subprograms that don't have Pre'Class just inherit whatever precondition the original subprogram has. Making an unspecified Pre'Class have the value True in this case would just cancel the original class-wide precondition, which is certainly not what we want.

The change to 6.1.1(18/3) is formally an inconsistency, as it will weaken (in fact, eliminate) the class-wide precondition that applies to statically bound calls (like call (3) in the question). This might be a problem if the body of the called subprogram (P in the example) assumes that the class-wide precondition is checked, and there were no dispatching calls in the program (which are already unchecked).

As such, we defined a Legality Rule to effectively change this inconsistency into an illegality (and incompatibility with the original definition of Ada 2012). Any such Pre'Class would have no effect (it would be "counterfeited") and would be dangerously misleading to readers.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0151**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **6.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0151 A prefixed view of a By_Protected_Procedure interface has convention protected

Working Reference Number AI12-0107-1

Question

A proposed ACATS B-Test contains a case like the following:

```
package Nested is
   type Intf is synchronized interface;
   procedure PPN1 (Param : in out Intf) is abstract
      with Synchronization => By_Protected_Procedure;
end Nested;

procedure Sink2 (P : access protected procedure) is
begin
   null; -- Eat an access-to-parameterless-protected-procedure.
end Sink2;

procedure Foo (Intf_In_Out_Parm : in out Nested.Intf'Class) is
begin
   Sink2(Intf_In_Out_Parm.PPN1'access);  -- (1)
end Foo;
```

The test expects (1) to be legal, but the convention of a prefixed view is intrinsic (by 6.3.1(10.1/2)), and that does not match the convention "protected" required by an access-to-protected-procedure.

The only special thing one can do with a protected procedure is to take 'Access of it for an access-to-protected-procedure type. It seems odd that this case does not work. Should it? (Yes.)

Summary of Response

A prefixed view of a subprogram with aspect Synchronization being By_Protected_Procedure has convention protected.

Corrigendum Wording

Replace 6.3.1(10.1):

- any prefixed view of a subprogram (see 4.1.3).

by:

- any prefixed view of a subprogram (see 4.1.3) without synchronization kind (see 9.5) By_Entry or By_Protected_Procedure.

Replace 6.3.1(12):

- The default calling convention is *protected* for a protected subprogram, and for an access-to-subprogram type with the reserved word **protected** in its definition.

by:

- The default calling convention is *protected* for a protected subprogram, a prefixed view of a subprogram with a synchronization kind of By_Protected_Procedure, and for an access-to-subprogram type with the reserved word **protected** in its definition.

Replace 6.3.1(13):

- The default calling convention is *entry* for an entry.

by:

- The default calling convention is *entry* for an entry and a prefixed view of a subprogram with a synchronization kind of By_Entry.

Discussion

The prefixes of interest are prefixed views where the subprogram has synchronization kind By_Protected_Procedure. (Aside: It's a bit odd that we don't have a matching By_Protected_Function synchronization kind, but that seems unnecessary for now.)

Such a prefixed view looks the same as a normal direct call to a protected procedure, and is known to be implemented by a protected procedure, so it would be unusual if it couldn't be used as one. After all, we do allow similar entry calls to be used in requeues (that was the reason for introducing synchronization kinds).

The question was raised in private mail whether a subprogram with By_Entry or By_Protected_Procedure should still have the Ada convention (thus, still being allowed as the prefix of Access for a *normal* access-to-procedure). (After all, this AI only changes the convention of prefixed views, not other uses.) This clearly would require a wrapper of some sort, but as it seems to be a normal subprogram for many other uses it's unclear that we need to prevent it. Thus we leave the convention of the subprogram itself as Ada. If we did want to prevent it, we could just declare such subprograms to have convention Intrinsic.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0152**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **6.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0152 Conformance of quantified expressions

Working Reference Number AI12-0050-1

Question

Consider the following example:

```
procedure Conform (S : String) is
   procedure Foo (X : Boolean :=
      (for all Index in S'range => (S (Index) /= '?')));
   procedure Foo (X : Boolean :=
      (for all Index in S'range => (S (Index) /= '?'))) is -- Legal??
   begin null; end Foo;
begin
   Foo;
end Conform;
```

Consider the two uses of the identifier Index. The first refers to the loop parameter declared in the first quantified expression; the second refers to the loop parameter declared in the second quantified expression. These are not the same, so the two default expressions are not fully conformant by 6.3.1(21).

This doesn't make any sense, is a fix needed? (Yes.)

Summary of Response

Quantified expressions fully conform if the loop parameter in each have the same identifier, and the range, iterable_name, or iterator_name fully conforms.

Corrigendum Wording

Insert after 6.3.1(20):

- each constituent construct of one corresponds to an instance of the same syntactic category in the other, except that an expanded name may correspond to a direct_name (or character_literal) or to a different expanded name in the other; and

the new paragraph:

- corresponding defining_identifiers occurring within the two expressions are the same; and

Replace 6.3.1(21):

- each direct_name, character_literal, and selector_name that is not part of the prefix of an expanded name in one denotes the same declaration as the corresponding direct_name, character_literal, or selector_name in the other; and

by:

- each direct_name, character_literal, and selector_name that is not part of the prefix of an expanded name in one denotes the same declaration as the corresponding direct_name, character_literal, or selector_name in the other, or they denote corresponding declarations occurring within the two expressions; and

Discussion

We talk about corresponding declarations because of the possibility of iterator_specifications (and parameterized_array_component_specifications -- see AI12-0061-1) occurring within default expressions.

We have to mention the defining identifiers being the same, so that (for all I1 in S'Range => (S (I1) /= '?'))) does not fully conform with (for all I2 in S'Range => (S (I2) /= '?'))) is -- Legal?? (No.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0153**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **6.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0153 Tag of the return object of a simple return expression

Working Reference Number AI12-0097-1

Question

The second sentence of 6.5(8/3) was modified by AI05-0032-1 to say:

"If the result type is class-wide, the tag of the return object is that of the type of the subtype_indication if it is specific, or otherwise that of the value of the expression."

Consider the case of a simple return statement (this rule applies to any return statement from a function). What "subtype_indication" are we talking about? There could be one somewhere in the expression (say in an allocator or slice), or even more than one, or there may not be one at all. This wording seems to imply that there is only one.

Probably this wording was crafted assuming that it only applies to an extended_return_statement, but that's not the case. Should the wording be clarified? (Yes.)

Summary of Response

The tag of a return object is that of the return type if that is specific. Otherwise, it is that of the return expression of a simple return statement, or the initializing expression of an extended return statement, unless that return statement has a subtype_indication of a specific type, in which case the tag of that type is used.

Corrigendum Wording

Replace 6.5(8):

> If the result type of a function is a specific tagged type, the tag of the return object is that of the result type. If the result type is class-wide, the tag of the return object is that of the type of the subtype_indication if it is specific, or otherwise that of the value of the expression. A check is made that the master of the type identified by the tag of the result includes the elaboration of the master that elaborated the function body. If this check fails, Program_Error is raised.

by:

> If the result type of a function is a specific tagged type, the tag of the return object is that of the result type. If the result type is class-wide, the tag of the return object is that of the value of the expression, unless the return object is defined by an extended_return_object_declaration with a subtype_indication that is specific, in which case it is that of the type of the subtype_indication. A check is made that the master of the type identified by the tag of the result includes the elaboration of the master that elaborated the function body. If this check fails, Program_Error is raised.

Discussion

Clearly, we have to cover the case of a simple_return_statement, and we do not want the effect of that to be different than it was in Ada 2005. So the wording has to make it clear that the only subtype_indication being talked about is the one directly part of an extended_return_statement.

Note that we use the relatively new syntax term extended_return_object_declaration in this wording, so there is no confusion about which subtype_indication we're talking about -- the current syntax of an extended_return_statement doesn't directly include any subtype_indication.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0154**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **7.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0154 Descendants of incomplete views

Working Reference Number AI12-0065-1

Question

The example of AARM 7.3.1(5.a.1-4/3) doesn't seem to have much to do with the paragraph 7.3.1(5.2/3) that precedes it. Moreover, that paragraph is supposed to be redundant, but it claims that types can be descendants of incomplete views of a type, which is not backed up by any other text in the standard.

Should this be clarified? (Yes.)

Summary of Response

It is possible for a type T3 to be indirectly derived from a type T1 but inherit no characteristics from T1, because one of its ancestors, say T2, is declared at a point where the derivation from T1 is not visible. The wording in 7.3.1 should be clarified to explain this.

Corrigendum Wording

Replace 7.3.1(5.2):

> It is possible for there to be places where a derived type is visibly a descendant of an ancestor type, but not a descendant of even a partial view of the ancestor type, because the parent of the derived type is not visibly a descendant of the ancestor. In this case, the derived type inherits no characteristics from that ancestor, but nevertheless is within the derivation class of the ancestor for the purposes of type conversion, the "covers" relationship, and matching against a formal derived type. In this case the derived type is considered to be a *descendant* of an incomplete view of the ancestor.

by:

> Furthermore, it is possible for there to be places where a derived type is known to be derived indirectly from an ancestor type, but is not a descendant of even a partial view of the ancestor type, because the parent of the derived type is not visibly a descendant of the ancestor. In this case, the derived type inherits no characteristics from that ancestor, but nevertheless is within the derivation class of the ancestor for the purposes of type conversion, the "covers" relationship, and matching against a formal derived type. In this case the derived type is effectively a *descendant* of an incomplete view of the ancestor.

Response

Paragraph 7.3.1(5.2/3) was intended to be a clarification of 7.3.1(5.1/3), but its wording was sufficiently confusing that it only made matters worse. We have tried to clarify the intent a bit.

Also, the AARM example was similarly intended to clarify things, but the mapping between the example and 5.2/3 was not clear, so again, things only became further muddled. Hopefully the added comments have helped explain how the example relates to 5.2/3.

Discussion

None needed.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0155**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0155 Type_Invariant'Class for interface types

Working Reference Number AI12-0041-1

Question

Type_Invariant is used on a private type to define constraints on the implementation. As such, they are particularly useful when derivation is used as a means to interact with a component. The developer of a component can use a combination of Post'Class and Type_Invariant'Class to ensure that the contracts are correctly fulfilled by the client.

One very common pattern to provide these interactions between components is to use interfaces, which are much more flexible than abstract tagged types. Unfortunately, Type_Invariant'Class cannot be specified on an interface, thus limiting the capability in this context. In particular, the invariant will allow one to express the relation between primitives and / or the type and its environment.

Summary of Response

Allow specifying Type_Invariant'Class for interface types.

Corrigendum Wording

Replace 7.3.2(1):

For a private type or private extension, the following language-defined aspects may be specified with an aspect_specification (see 13.1.1):

by:

For a private type, private extension, or interface, the following language-defined aspects may be specified with an aspect_specification (see 13.1.1):

Replace 7.3.2(3):

Type_Invariant'Class
This aspect shall be specified by an expression, called an *invariant expression*. Type_Invariant'Class may be specified on a private_type_declaration or a private_extension_declaration.

by:

Type_Invariant'Class
This aspect shall be specified by an expression, called an *invariant expression*. Type_Invariant'Class may be specified on a private_type_declaration, a private_extension_declaration, or a full_type_declaration for an interface type.

Response

Allow Type_Invariant'Class on interfaces. When a type implements one or several interfaces, its inherited type invariant would then be the conjunction of all ancestor Type_Invariant'Class.

Discussion

The existing Dynamic Semantics that describes invariant checks works perfectly for interfaces; it does not need any changes.

Note that an abstract null record is very similar to an interface. We could allow them with no other changes. We didn't do this as it doesn't seem to add much to the language, and allowing components would cause problems (see next paragraph).

A bigger alternative would be to allow invariants on most non-private types. This idea was rejected during the development of Ada 2012 as it would allow invariants to be violated almost anywhere, rather than only within the defining package. (The protection offered by invariants is not quite complete, but it is close, and the

programmer of the package can easily create abstractions that have no holes.) There is no good reason to revisit this decision (particularly so early in the life of Ada 2012 - the standard has been approved only for a few weeks as this is written).

Note that the problem of AI12-0042-1 appears to be related to this AI, but it isn't really related as it can happen whether or not interfaces allow invariants. The solution adopted for that AI will work for interfaces.

Example

-- This window should always display exactly 100 pixels

```
type Window is interface
   with Type_Invariant'Class => Window.Get_Width * Window.Get_Height = 100;

function Get_Width (This : Window) return Integer is abstract;
function Get_Height (This : Window) return Integer is abstract;
```

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0156**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0156 Class-wide type invariants and statically bound calls

Working Reference Number AI12-0150-1

Question

RM 7.3.2(5/3) says:

[AARM Redundant: Within an invariant expression, the identifier of the first subtype of the associated type denotes the current instance of the type.] Within an invariant expression associated with type T, the type of the current instance is T for the Type_Invariant aspect and T'Class for the Type_Invariant'Class aspect.

However, in analogy with Pre'Class as discussed in AI12-0113-1, simply resolving all calls appearing in the Type_Invariant'Class expression for a type T by interpreting it as T'Class will cause all calls to be dispatching calls, even when applied to objects where their compile-time type does not match the underlying tag were it to be converted to T'Class.

Should this be corrected? (Yes.)

Summary of Response

Class-wide preconditions, postconditions, and type invariants are processed using the actual types for the parameters of type T (not always using T'Class). In particular, for a statically bound call, the calls within the Type_Invariant'Class expression that apply to the type T are statically bound, if the object being checked is itself of a specific type.

Corrigendum Wording

Replace 7.3.2(3):

Type_Invariant'Class
This aspect shall be specified by an expression, called an *invariant expression*. Type_Invariant'Class may be specified on a private_type_declaration or a private_extension_declaration.

by:

Type_Invariant'Class
This aspect shall be specified by an expression, called an *invariant expression*. Type_Invariant'Class may be specified on a private_type_declaration, a private_extension_declaration, or a full_type_declaration for an interface type. Type_Invariant'Class determines a *class-wide type invariant* for a tagged type.

Replace 7.3.2(5):

Within an invariant expression, the identifier of the first subtype of the associated type denotes the current instance of the type. Within an invariant expression associated with type *T*, the type of the current instance is *T* for the Type_Invariant aspect and *T*'Class for the Type_Invariant'Class aspect.

by:

Within an invariant expression, the identifier of the first subtype of the associated type denotes the current instance of the type. Within an invariant expression for the Type_Invariant aspect of a type *T*, the type of this current instance is *T*. Within an invariant expression for the Type_Invariant'Class aspect of a type *T*, the type of this current instance is interpreted as though it had a (notional) type *NT* that is a visible formal derived type whose ancestor type is *T*. The effect of this interpretation is that the only operations that can be applied to this current instance are those defined for such a formal derived type. This ensures that the invariant expression is well-defined for any type descended from *T*.

Replace 7.3.2(9):

If one or more invariant expressions apply to a type *T*, then an invariant check is performed at the following places, on the specified object(s):

by:

If one or more invariant expressions apply to a nonabstract type *T*, then an invariant check is performed at the following places, on the specified object(s):

Insert after 7.3.2(22):

The invariant check consists of the evaluation of each enabled invariant expression that applies to *T*, on each of the objects specified above. If any of these evaluate to False, Assertions.Assertion_Error is raised at the point of the object initialization, conversion, or call. If a given call requires more than one evaluation of an invariant expression, either for multiple objects of a single type or for multiple types with invariants, the evaluations are performed in an arbitrary order, and if one of them evaluates to False, it is not specified whether the others are evaluated. Any invariant check is performed prior to copying back any by-copy **in out** or **out** parameters. Invariant checks, any postcondition check, and any constraint or predicate checks associated with **in out** or **out** parameters are performed in an arbitrary order.

the new paragraph:

For an invariant check on a value of type *T1* based on a class-wide invariant expression inherited from an ancestor type *T*, any operations within the invariant expression that were resolved as primitive operations of the (notional) formal derived type *NT*, are in the evaluation of the invariant expression for the check on *T1* bound to the corresponding operations of type *T1*.

Discussion

Consider two nearly identical hierarchies of types. First:

```
package P11 is
    type Root is abstract tagged private
      with Type_Invariant => Is_Valid (Root);

    function Is_Valid (P : in Root) return Boolean;

    procedure Proc (P : out Root);

    procedure Proc2 (P : out Root);
private
    ...
end P11;

with P11;
package P12 is
    type Child is new P11.Root with private
      with Type_Invariant => Is_Valid (Child);

    overriding
    function Is_Valid (P : in Child) return Boolean;

    overriding
    procedure Proc (P : out Child);

    -- Proc2 inherited.
private
    ...
end P12;

with P12;
package P13 is
    type GrandChild is new P12.Child with private
      with Type_Invariant => Is_Valid (GrandChild);

    overriding
    function Is_Valid (P : in Grandchild) return Boolean;

    overriding
    procedure Proc (P : out Grandchild);

    -- Proc2 inherited.
private
    ...
```

```
end P13;
```

and the second hierarchy is:

```
package P21 is
    type Root is abstract tagged private
      with Type_Invariant'Class => Is_Valid (Root);

    function Is_Valid (P : in Root) return Boolean;

    procedure Proc (P : out Root);

    procedure Proc2 (P : out Root);
private
    ...
end P21;

with P21;
package P22 is
    type Child is new P21.Root with private;

    overriding
    function Is_Valid (P : in Child) return Boolean;

    overriding
    procedure Proc (P : out Child);

    -- Proc2 inherited.
private
    ...
end P22;

with P22;
package P23 is
    type GrandChild is new P22.Child with private;

    overriding
    function Is_Valid (P : in Grandchild) return Boolean;

    overriding
    procedure Proc (P : out Grandchild);

    -- Proc2 inherited.
private
    ...
end P23;
```

The only difference between these hierarchies is that the first uses a specific Type_Invariant on each type, the second uses a class-wide Type_Invariant just on the Root type, allowing it to be inherited by descendant types.

Our intent is that these behave identically at runtime, with the only difference being that there is an additional guarantee that a new descendant of P21.Root will also have the Type_Invariant'Class expression (that has to be manually added in the other hierarchy).

We define the meaning of Type_Invariant'Class in terms of a notional formal derived type so that only primitive operations of the type can be used in the expression. The original Ada 2012 definition of the current instance as T'Class also has a runtime effect that we do not want (as noted above).

In addition, defining the current instance as having T'Class introduces other problems. For instance, with a global variable: type T is tagged ... with Type_Invariant'Class => Is_Valid_2 (T, G); G : T'Class := ...; function Is_Valid_2 (A, B : T) return Boolean;

If the current instance T is considered to have T'Class, this expression is legal (both operands being dynamically tagged). But this doesn't make sense for an inherited type: type T1 is new T with ...; -- with Type_Invariant'Class => Is_Valid_2 (T, G) -- inherits Is_Valid_2 (A, B : T1) return Boolean;

But this Type_Invariant'Class makes no sense, as G does not match type T1'Class.

If we simply left the current instance having type T, then non-primitive operations of the type could be used in Type_Invariant'Class -- but descendant types have no such operations. We could have rechecked the Type_Invariant'Class for each descendant type, but that causes problems if visibility has changed somehow (especially for class-wide operations).

On the other hand, a formal derived type has just the operations that we want to allow (primitive operations) but no others. In particular, G in the example above does not match the NT type, so the above problem can't happen.

We added "nonabstract" to 7.3.2(9/3) because a class-wide invariant cannot be reliably enforced on an abstract type, because some of the operations used in the original invariant expression might be abstract for an abstract descendant. Even if all of the operations are nonabstract for a particular abstract descendant, it seems unwise and a potential maintenance trap to have the enforcement of the class-wide invariant depend on that.

This does not provide a hole in the checking provided by class-wide type invariants, even in the case of a concrete operation inherited by a descendant, because the checking on the implicit view conversions that are part of a call on an inherited subprogram will make the checks for any object of a concrete type. (And there cannot be an object of abstract type!)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0157**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0157 Type invariant checking rules

Working Reference Number AI12-0042-1

Question

If a Type_Invariant'Class is specified for a type, which operations of a descendant type must perform the invariant check when the descendant type is not a private type or private extension?

Summary of Response

If a class-wide invariant applies to an ancestor, then any private operation of the ancestor type that is visible at the point of the extension shall be overridden. In addition, if a type invariant is inherited by a record extension, the type invariant is checked after any operation that corresponds to a visible operation of an ancestor to which this invariant applies.

Corrigendum Wording

Insert after 7.3.2(6):

The Type_Invariant'Class aspect shall not be specified for an untagged type. The Type_Invariant aspect shall not be specified for an abstract type.

the new paragraph:

If a type extension occurs at a point where a private operation of some ancestor is visible and inherited, and a Type_Invariant'Class expression applies to that ancestor, then the inherited operation shall be abstract or shall be overridden.

Replace 7.3.2(17):

- is declared within the immediate scope of type *T* (or by an instance of a generic unit, and the generic is declared within the immediate scope of type *T*), and

by:

- is declared within the immediate scope of type *T* (or by an instance of a generic unit, and the generic is declared within the immediate scope of type *T*),

Delete 7.3.2(18):

- is visible outside the immediate scope of type *T* or overrides an operation that is visible outside the immediate scope of *T*, and

Replace 7.3.2(19):

- has a result with a part of type *T*, or one or more parameters with a part of type *T*, or an access to variable parameter whose designated type has a part of type *T*.

by:

- has a result with a part of type *T*, or one or more parameters with a part of type *T*, or an access to variable parameter whose designated type has a part of type *T*;
- and either:
  - *T* is a private type or a private extension and the subprogram or entry is visible outside the immediate scope of type *T* or overrides an inherited operation that is visible outside the immediate scope of *T*, or
  - *T* is a record extension, and the subprogram or entry is a primitive operation visible outside the immediate scope of type *T* or overrides an inherited operation that is visible outside the immediate scope of *T*.

Insert after 7.3.2(20):

The check is performed on each such part of type *T*.

the new paragraph:

- For a view conversion to a class-wide type occurring within the immediate scope of *T*, from a specific type that is a descendant of *T* (including *T* itself), a check is performed on the part of the object that is of type *T*.

Discussion

This AI addresses three issues:

1) If a type extension inherits from some ancestor both a Type_Invariant'Class and a private operation, then we've added a rule that the operation must be either overridden or abstract. The point is that the class-wide Type_Invariant of the ancestor didn't apply to the original operation (because it was a private operation) but it applies to the inherited operation. (An example of such a case is given below.)

In such a case, the private operation would not be expected to maintain the invariant (as it is inside the checking boundary). However, the inherited routine would be required to maintain the invariant (as it is now on the checking boundary). We require overriding (or abstractness) in the case of inherited subprograms that have different contracts than the "original" ancestor subprogram when that significantly changes the meaning of the routine (this includes preconditions as well as type invariants).

This is just to avoid surprising behavior, not because of any real definitional problem. It also spares implementations from having to generate wrapper routines in this case.

2) In 7.3.2(18/3), the existing wording:

is visible outside the immediate scope of type T or overrides an operation that is visible outside the immediate scope of T

was wrong in two ways and needed to be fixed up.

First, this wording didn't correctly address the case of a record extension.

Second, the existing wording talked about overriding operations of a parent type. But a derived type never overrides operations of its parent type - instead, it may override implicitly-declared operations that were inherited from the parent type.

3) Class-wide objects, which could represent a hole in the checking mechanism.

We considered the more general issue of invariants that apply to record extensions. This can happen two ways. One is a Type_Invariant'Class inherited into a record extension. Similarly, invariants can be added to private extensions of record types that have visible components. The checking for type invariants was designed to catch virtually all cases where the objects cross the package boundaries. When there are visible components, this model breaks down as the visible components can be modified independently of the package boundaries, which could make the invariant False without detection. Both cases could be prevented with Legality Rules (as we do not allow class-wide invariants to be hidden). We decided it's not worth preventing such things, even with the possibility of misuse.

Similarly, there are other holes in the checking represented by type invariants (some of these are explained in AARM 7.3.2(20.a/3)). We do not believe the effort to plug all possible holes is practical. As such, we only plug holes that are likely to occur in practice (like the use of class-wide objects).

Example

An example of item (1) from the discussion:

```
package Type_Invariant_Pkg is
   type T1 is tagged private with Type_Invariant'Class => Is_Ok (T1);
   function Is_Ok (X1 : T1) return Boolean;
   procedure Op (X : in out T1); -- [1]
```

```
private
   type T1 is tagged record F1, F2 : Natural := 0; end record;
   function Is_Ok (X1 : T1) return Boolean is (X1.F1 <= X1.F2);
   procedure Priv_Op (X : in out T1); -- [2]
end Type_Invariant_Pkg;

private package Type_Invariant_Pkg.Child is
   type T2 is new T1 with null record; -- Illegal by this AI.
   -- Inherits procedure Op (X : in out T2); -- [3]
   -- Inherits procedure Priv_Op (X : in out T2); -- [4]
end;
```

Note that a call on Op [1] will cause the invariant to be checked on return, while a call on Priv_Op [2] would not make such an invariant check.

However, a call on either Op [3] or Priv_Op [4] will cause the invariant to be checked on return. It's this change of the check that applies to Priv_Op that makes this illegal.

Note that this case is rather unlikely. If the keyword private is erased from Type_Invariant_Pkg.Child, then the error goes away (as Priv_Op [4] would be declared in the private part of Type_Invariant_Pkg.Child and no invariant check would be required).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0158**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0158 Invariants need to be checked on the initialization of deferred constants

Working Reference Number AI12-0049-1

Question

Consider:

```
package R is
   type T is private with Type_Invariant => Non_Null (T);
   function Non_Null (X : T) return Boolean;
   Zero : constant T;
private
   type T is new Integer;
   function Non_Null (X : T) return Boolean is (X /= 0);
   Zero : constant T := 0;
end R;
```

Zero violates the invariant, but this will not be detected because there is no rule in the Standard that says this should be checked.

Should this be fixed? (Yes.)

Summary of Response

Invariants are checked on the initialization of deferred constants.

Corrigendum Wording

Insert after 7.3.2(10):

- After successful default initialization of an object of type *T*, the check is performed on the new object;

the new paragraph:

- After successful explicit initialization of the completion of a deferred constant with a part of type *T*, if the completion is inside the immediate scope of the full view of *T*, and the deferred constant is visible outside the immediate scope of *T*, the check is performed on the part(s) of type *T*;

Discussion

We want every value that can "escape" from the visible part of the package to have the invariant checked. There certainly shouldn't be any easy end runs around the checking.

This wording seems more complicated than necessary, but a number of considerations arise.

First, we have to talk about "the completion of a deferred constant" as the deferred constant itself has no initializer (and thus nothing to check).

Second, we have to check more than just deferred constants of type T. Deferred constants can be of any type, and thus T could have been used as a component:

```
package R2 is
   type T is private with Type_Invariant => Non_Null (T);
   function Non_Null (X : T) return Boolean;
   type Conditional_T is record
       Valid : Boolean;
       Data  : T;
   end record;
   Invalid : constant Conditional_T;
private
```

```
   type T is new Integer;
   function Non_Null (X : T) return Boolean is (X /= 0);
   Invalid : constant Conditional_T := (Valid => False, Data => 0);
end R2;
```

This has the same leak as the original example. Thus, we have to talk about "a part of type T".

Third, we have to also make the check in visible child packages. Consider this child of the original example:

```
package R.C is
   Zero : constant T;
private
   Zero : constant T := 0;
end R.C;
```

This also has the same hole as in the original example. The wording here talks about "immediate scope", so that the check does not apply to private child packages (which cannot leak values out of the "subsystem").

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0159**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0159 Type invariants and default initialized objects

Working Reference Number AI12-0133-1

Question

7.3.2(10/3) talks of "default initialization" which isn't obviously defined (it's not in the index of the Standard). Was "initialized by default" intended? (Yes.)

Typically, invariant checks are not made on objects declared within the package body. But 7.3.2(10/3) has no such exception. That means that temporary objects and <> in aggregates are at risk of Assertion_Error even in the body. Is this intended? (Yes.)

Summary of Response

```
type invariants are checked on all objects of the type that are initialized
by default.
```

Corrigendum Wording

Replace 7.3.2(10):

- After successful default initialization of an object of type *T*, the check is performed on the new object;

by:

- After successful initialization of an object of type *T* by default (see 3.3.1), the check is performed on the new object unless the partial view of *T* has unknown discriminants;

Discussion

The term "default initialization" is not defined. It makes more sense to to use a defined term. That ensures that it applies to all objects initialized by default (including <> components of aggregates, components of larger objects, stand-alone objects, etc.).

The author had some concern about the ordering rules that get dragged along with "initialized by default". However, those aren't a problem here, because this wording only applies "after initialization", and all of the ordering issues will have been sorted out first.

For the second question, normally it does not matter whether or not it is checked within the package. Default initialization of a type is the same inside or outside of a unit, so it needs to pass the checks no matter where it occurs. As such, the rule is simplified by having it apply everywhere.

However, if the private type with a type invariant has unknown discriminants, then no default initialization is allowed outside the package. In that case, checking the invariant could only happen inside the package, which makes no sense (and could make it difficult to construct objects as default initialized objects and components could raise Assertion_Error). Therefore, we add an exception specifically for this case.

Example of possible issues if a type with unknown discriminants is checked:

```
package P is
   type T (<>) is private;
   function Make (Val : Integer) return T;
private
   type T is record
      A : access Integer := null;
   end record
      with Type_Invariant => Is_OK(T);
   function Is_OK (Obj : T) return Boolean is
       (Obj.A /= null);
```

```
end P;

package body P is
   -- Declare a linked list node:
   type Node;
   type Ptr is access Node;
   type Node is record
      C : T;
      N : Ptr;
   end record;

   function Make (Val : Integer) return T is
   begin
      if Val = 1 then
         declare
            Temp : T; -- No check here. If there was one, the check would
                      -- fail and raise Assertion_Error during default init.
         begin
            Temp.A := new Integer'(1);
            return Temp;
         end;
      elsif Val = 2 then
         return Foo : T := (A => <>) do -- OK, no exception whether or not
                                        -- a check is made.
            Foo.A := new Integer'(Val);
         end return;
      else
         -- The following structure is likely if there is a linked list
         -- in the body (although it would probably be in some other
         -- subprogram):
         declare
             N : Node := (C => <>, N => null); -- No check here. If there was
                 -- one, the check would fail and raise Assertion_Error.
         begin
             N.C.A := new Integer'(Val);
             return N.C;
         end;
      end if;
   end Make;
end P;
```

Neither of the places marked "no check here" should have a check, since they're inside of the package and are just constructing values to return to the client.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0160**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0160 Should say stream-oriented attribute

Working Reference Number AI12-0146-1

Question

7.3.2(15/3) says "After a successful call on the Read or Input stream attribute of the type T, the check is performed on the object initialized by the stream attribute;".

However, there is no such thing (formally) as a "stream attribute". The title of 13.13.2 is "Stream-oriented attributes"; there isn't even an index entry for "stream attribute".

Should we change this wording? (Yes.)

Summary of Response

Read and Input are stream-oriented attributes.

Corrigendum Wording

Replace 7.3.2(15):

- After a successful call on the Read or Input stream attribute of the type *T*, the check is performed on the object initialized by the stream attribute;

by:

- After a successful call on the Read or Input stream-oriented attribute of the type *T*, the check is performed on the object initialized by the attribute;

Discussion

None needed.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0161**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0161 Calling visible functions from type invariant expressions

Working Reference Number AI12-0044-1

Question

AI05-0289-1 extends invariant checking to "in" parameters. However, this makes it impossible to call a public function of the type from an invariant expression, as that public function will attempt to check the invariant, resulting in infinite recursion.

For Type_Invariant'Class, which is required to be visible, it is almost impossible to write a useful invariant without calling a public function.

Is it intended that invariants be useless? (No.)

Summary of Response

The type invariants of IN parameters are checked on exit from procedures, but not from functions, when the Assertion_Policy is Check. (Implementations can always provide other policies which provide more rigorous checking of IN parameters.)

Corrigendum Wording

Replace 7.3.2(19):

- has a result with a part of type *T*, or one or more parameters with a part of type *T*, or an access to variable parameter whose designated type has a part of type *T*.

by:

- has a result with a part of type *T*, or one or more **out** or **in out** parameters with a part of type *T*, or an access-to-object parameter whose designated type has a part of type *T*;
- is a procedure or entry and has an **in** parameter with a part of type *T*.

Discussion

The easiest fix to this problem would be to revert to not checking "in" parameters (that is, to repeal AI05-0289-1). However, this does not address any of the issues that caused AI05-0289 to be approved in the first place.

We considered many approaches to fixing this problem, but they all added complexity to almost any use of Type_Invariant. Limiting the checks on IN parameters to procedures is simple to explain and implement, and avoids portability issues associated with bounded errors or implementation permissions. Clearly plenty of holes remain, so the goal is to reduce the number of holes without adding complexity for the user.

The long-term solution of this issue depends on providing aspects that indicate when global (or indirectly-referenced) state is modified, and we don't want to interfere with an appropriate solution to that problem by overly constraining the problem now with quick-fix special-case aspects.

Note that we changed "access-to-variable parameters" to be simply "access parameters," as access-to-constant parameters cannot contribute to the problem of infinite recursion, but can represent a hole in exactly the same way an IN parameter can represent a hole, so there seems no reason to limit the checks to only access-to-variable parameters.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0162**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **7.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0162 Type invariants are checked for functions returning access-to-type

Working Reference Number AI12-0149-1

Question

7.3.2(19.3/3) says that the invariant for a type T is checked for parameters of an access-to-object-containing-a-part-of-T. But there is not corresponding rules for functions that return access types. That seems to create a significant hole in the protection provided by type invariants.

Should this hole be plugged? (Yes.)

Summary of Response

```
type invariants are checked for functions whose result is access to
a type with a part of a type with an invariant.
```

Corrigendum Wording

Replace 7.3.2(19):

- has an access-to-object parameter whose designated type has a part of type *T*, or

by:

- has an access-to-object parameter or result whose designated type has a part of type *T*, or

@drep

Discussion

As noted in the AARM note above, it is intended that type invariant checks aren't made on composite types with subcomponents of an access-to-object-with-part-of-T, only on direct access-to-object-with-part-of-T. It's not practical to check all such cases, as one would have to look through all of the access-to-somethings to do so.

Specifically, if we made the rule stronger by including parts of access-to-object types:

* has a parameter or result with a part of an access-to-object type whose designated type has a part of type T, or

There would still be an unchecked case for an access-to type with a part of a access-to part of T. And these cases are getting increasingly unlikely.

In any case, since the checks performed are determined by the declarations in the visible part of the package containing T, the author of the package containing T has complete control over whether any holes exist. So long as they do not define a visible type with a subcomponent that is access-to-object-with-part-of-T (and do not export any private subprograms via an access-to-subprogram type), all possible ways for a value to leave the package will be checked. (There is no problem if a private type contains a subcomponent of an access type, as any subprogram that provides access to that component will necessarily have to have a parameter or result that is itself checked.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0163**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **8.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0163 An access_definition should be a declarative region

Working Reference Number AI12-0094-1

Question

Is the following legal?

```
type T is
 access function (A : Integer) return
 access function (A : Float) return Boolean;
```

One might expect that it is legal, but according to RM "8.1 Declarative Region" there is a declarative region for T and two declarations of A immediately within it, which is illegal by 8.3(26/2).

Should we change the definition of "declarative region" to include an access_definition? (Yes.)

Summary of Response

An access_definition is a declarative region.

Corrigendum Wording

Insert after 8.1(2):

- any declaration, other than that of an enumeration type, that is not a completion of a previous declaration;

the new paragraph:

- an access_definition;

Discussion

This clearly seems like an oversight.

GNAT allows this example (it thinks it is legal).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0164**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **8.2; 13.11.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0164 Specifying the standard storage pool

Working Reference Number AI12-0003-1

Question

If a programmer wants to specify the default storage pool as NULL through a configuration pragma, then how would one provide a user defined storage pool that does allocate a block of memory from the heap (aka the Standard storage pool as described in 13.11)? (Use storage_pool_indicator Standard.)

Summary of Response

The standard storage pool can be specified in a Default_Storage_Pool pragma or aspect.

Corrigendum Wording

Insert after 8.2(11):

The immediate scope of a declaration is also the immediate scope of the entity or view declared by the declaration. Similarly, the scope of a declaration is also the scope of the entity or view declared by the declaration.

the new paragraph:

The immediate scope of a pragma that is not used as a configuration pragma is defined to be the region extending from immediately after the pragma to the end of the declarative region immediately enclosing the pragma.

Replace 13.11.3(1):

by:

Pragma and aspect Default_Storage_Pool specify the storage pool that will be used in the absence of an explicit specification of a storage pool or storage size for an access type.

Replace 13.11.3(3.1):

storage_pool_indicator ::= *storage_pool_*name | **null**

by:

storage_pool_indicator ::= *storage_pool_*name | **null** | Standard

Insert after 13.11.3(4):

The *storage_pool_*name shall denote a variable.

the new paragraph:

The Standard storage_pool_indicator is an identifier specific to a pragma (see 2.8) and does not denote any declaration. If the storage_pool_indicator is Standard, then there shall not be a declaration with defining_identifier Standard that is immediately visible at the point of the pragma, other than package Standard itself.

Replace 13.11.3(4.1):

If the pragma is used as a configuration pragma, the storage_pool_indicator shall be **null**, and it defines the *default pool* to be **null** within all applicable compilation units (see 10.1.5), except within the immediate scope of another pragma Default_Storage_Pool. Otherwise, the pragma occurs immediately within a sequence of declarations, and it defines the default pool within the immediate scope of the pragma to be either **null** or the pool denoted by the *storage_pool_*name, except within the immediate scope of a later pragma Default_Storage_Pool. Thus, an inner pragma overrides an outer one.

by:

If the pragma is used as a configuration pragma, the storage_pool_indicator shall be either **null** or Standard, and it defines the *default pool* to be the given storage_pool_indicator within all applicable compilation units (see 10.1.5), except within the immediate scope of another pragma

Default_Storage_Pool. Otherwise, the pragma occurs immediately within a sequence of declarations, and it defines the default pool within the immediate scope of the pragma to be the given storage_pool_indicator, except within the immediate scope of a later pragma Default_Storage_Pool. Thus, an inner pragma overrides an outer one.

Replace 13.11.3(5):

The language-defined aspect Default_Storage_Pool may be specified for a generic instance; it defines the default pool for access types within an instance. The expected type for the Default_Storage_Pool aspect is Root_Storage_Pool'Class. The aspect_definition must be a name that denotes a variable. This aspect overrides any Default_Storage_Pool pragma that might apply to the generic unit; if the aspect is not specified, the default pool of the instance is that defined for the generic unit.

by:

The language-defined aspect Default_Storage_Pool may be specified for a generic instance; it defines the default pool for access types within an instance.

The Default_Storage_Pool aspect may be specified as Standard, which is an identifier specific to an aspect (see 13.1.1) and defines the default pool to be Standard. In this case, there shall not be a declaration with defining_identifier Standard that is immediately visible at the point of the aspect specification, other than package Standard itself.

Otherwise, the expected type for the Default_Storage_Pool aspect is Root_Storage_Pool'Class and the aspect_definition shall be a name that denotes a variable. This aspect overrides any Default_Storage_Pool pragma that might apply to the generic unit; if the aspect is not specified, the default pool of the instance is that defined for the generic unit.

Replace 13.11.3(6.2):

- If the default pool is nonnull, the Storage_Pool attribute is that pool.

by:

- If the default pool is neither **null** nor Standard, the Storage_Pool attribute is that pool.

Replace 13.11.3(6.3):

Otherwise, there is no default pool; the standard storage pool is used for the type as described in 13.11.

by:

Otherwise (including when the default pool is specified as Standard), the standard storage pool is used for the type as described in 13.11.

Response

Add a new argument to the Default_Storage_Pool pragma to achieve the effect of returning to the "standard" allocation method used by the implementation, usually the use of a general heap.

Discussion

We do not want to give the standard storage pool a name, since it should remain implementation-defined whether, in the absence of user-provided storage pools, the storage pool model is used by the implementation at all.

Specifically it might just go for the heap, or have a clever way of combining a general heap with pools of cached, equally-sized memory blocks.

Extending the Default_Storage_Pool pragma to allow it to cause the implementation to revert to the standard allocation method seemed best to achieve the desired capability.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0165**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **8.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0165 Resolving the selecting_expression of a case_expression

Working Reference Number AI12-0040-1

Question

The selecting_expression of a case_statement is a complete context by 8.6(9); the selecting_expression of a case_expression is not. In unusual cases, this can make case_expressions legal where the corresponding case_statement would be illegal.

Consider:

```
procedure Complete_Contexts is
  type E1 is (Aa, Bb, Cc);
  type E2 is (Bb, Cc, Dd);

  function F return E1 is begin return E1'First; end F;
  function F return E2 is begin return E2'First; end F;

  N : Natural;
begin
  N := (case F is -- Legal? (No.)
          when Aa => 123,
          when Bb => 456,
          when Cc => 789);

  case F is -- Illegal.
     when Aa => null;
     when Bb => null;
     when Cc => null;
  end case;
end;
```

Should this be fixed? (Yes.)

Summary of Response

The selecting_expression of a case_expression resolves in the same way as the selecting_expression of a case_statement; in particular, it is a complete context.

Corrigendum Wording

Replace 8.6(9):

- The expression of a case_statement.

by:

- The *selecting_*expression of a case_statement or case_expression.

Discussion

We want case statements and case expressions to resolve the same way; consistency is important.

It's a bit odd to have a complete context in the middle of an expression, but as the wording of 8.6(4) makes it clear that complete contexts can be nested, there is no semantic problem. Moreover, other constructs like the operand of a type conversion and conditions of an if expression act like complete contexts (their resolution has no effect on whether the rest of the expression can be resolved). So this really isn't new.

A selecting_expression has to be a complete context in order to prevent the case choices from affecting the resolution. In cases that aren't complete contexts (like the target of an assignment), all of the possibilities are considered together (thus the source expression of an assignment can help the resolution of the target).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0166**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **8.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0166 Predicates and the current instance of a subtype

Working Reference Number AI12-0068-1

Question

For a predicate, is the current instance of the subtype an object or a value? This matters in unusual cases. Consider:

```
type T (Kind : T_Kind := Zoofle) is ... ;

-- A variable is in this subtype if and only if it
-- is ok to assign it a value whose Kind is Zoofle
subtype Zoofleable is T with
  Dynamic_Predicate =>
    Zoofleable.Kind = Zoofle
    or else not Zoofleable'Constrained;

procedure Zoofalize (X : out Zoofleable) is
begin
    X := (Kind => Zoofle, ...);
    -- Assignment's discriminant check guaranteed to pass
end;

Y : T := (Foozle, ...); -- unconstrained, Kind /= Zoofle
Z : T (Foozle) := Y; -- constrained, Kind /= Zoofle
begin
Zoofalize (Y); -- Works
Zoofalize (Z); -- Fails predicate check (??)
```

Is it expected that the value of 'Constrained reflects that of the actual object, or is this check on a value and thus the properties of the object can be queried? (Can't be queried.)

In a related question, is the nominal subtype of the current instance of a type or subtype defined? This matters in case expressions. Given this example,

```
subtype S is Natural with Dynamic_Predicate => (case S is ...
```

is coverage of the negative values required or forbidden? (Forbidden.)

Summary of Response

The current instance of a subtype acts like a value of the type, not as an object.

Corrigendum Wording

Insert after 8.6(17):

- If a usage name appears within the declarative region of a type_declaration and denotes that same type_declaration, then it denotes the *current instance* of the type (rather than the type itself); the current instance of a type is the object or value of the type that is associated with the execution that evaluates the usage name. Similarly, if a usage name appears within the declarative region of a subtype_declaration and denotes that same subtype_declaration, then it denotes the current instance of the subtype. These rules do not apply if the usage name appears within the subtype_mark of an access_definition for an access-to-object type, or within the subtype of a parameter or result of an access-to-subprogram type.

the new paragraph:

Within an aspect_specification for a type or subtype, the current instance represents a value of the type; it is not an object. The nominal subtype of this value is given by the subtype itself (the first subtype in the case of a type_declaration), prior to applying any predicate specified directly on the type or subtype. If the type or subtype is by-reference, the associated object with the value is the object associated (see 6.2) with the execution of the usage name.

Response

(See !summary.)

Discussion

Note that at the moment, a current instance of a type and a subtype is a variable, since it is not in the list of things that denote constant views in 3.3. That list is supposed to be exhaustive (and we keep adding to it as things are missed). Moreover, it is clear that it is a variable by practice, as the Rosen trick depends on that fact:

```
type Outer;
type Inner (Ref : access Outer) is limited null record;
  -- discriminant's type is access-to-variable

type Outer is limited record
  Self : Inner (Outer'access);
end record;
```

This would not be legal if Outer was a constant view (as converting an access-to-constant value to an access-to-variable value is illegal).

We considered the model that the current instance of a subtype acts like an in-mode parameter for the purposes of use in a predicate definition. This is the most information that a predicate should be allowed to use, as we want to allow compilers to implement a predicate as a Boolean function with a parameter of the base type of the subtype.

But we chose to treat it as a constant value. By choosing this model, even using object attributes within a predicate are illegal. For instance, 'Size, 'Alignment, 'Access, and 'Address are all illegal as their prefix does not denote an object.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0167**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **9.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0167 Expression functions and null procedures can be declared in a protected_body

Working Reference Number AI12-0147-1

Question

Is it permissible to use an expression function as the completion of a protected function? (Yes.)

6.8(2/3) defines an expression_function_declaration as something separate from a subprogram_declaration, but 9.4(8/1) only allows subprogram_declaration, subprogram_body, entry_body, and aspect_clause.

Summary of Response

Expression functions and null procedures can be declared in the body of a protected type.

Corrigendum Wording

Replace 9.4(8):

```
protected_operation_item ::= subprogram_declaration
    | subprogram_body
    | entry_body
    | aspect_clause
```

by:

```
protected_operation_item ::= subprogram_declaration
    | subprogram_body
    | null_procedure_declaration
    | expression_function_declaration
    | entry_body
    | aspect_clause
```

Discussion

It appears to be an oversight that expression functions are not allowed in a protected body. They are semantically the same as the full declaration of a function body with a single return statement, so there can be no significant implementation burden, and there seems to be no reason to not allow the shorthand in protected bodies.

Similarly, null procedures should be allowed in protected bodies. They are also semantically the same as a null body for a procedure, and again there is no reason to not allow the shorthand.

Note that this not only allows expression functions and null procedures to be completions, but also to declare body-only expression functions and null procedures. The language already allows that for subprogram declarations, and while a hidden null procedure doesn't seem useful, a hidden function can be used to encapsulate a complex barrier expression. There seems to be no reason to require a full body for such a function rather than allowing an expression function.

Note that we are *not* allowing expression functions or null procedures to be used in the specification of a protected type or object. These contexts do not currently allow any sort of body, and there may be implementation complications in allowing that. That seems OK, as the specification of a protected_type is a very restricted place as to what can be written, so the apparent inconsistency with package specifications is insignificant - you can't have types (anywhere) or objects (in the visible part) of a protected_type, either.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0168**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **9.5.1; 9.5.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0168 Make protected objects more protecting

Working Reference Number AI12-0129-1

Question

The design of containers assumes that they are not accessed concurrently, on the ground that access can be protected with protected objects if necessary.

However, containers provide functions that modify the state of the container; it is therefore not possible to access these through protected functions, as these can be called concurrently. Should there be a way to allow this? (Yes.)

Summary of Response

A Boolean aspect Exclusive_Functions is added to the language.

Corrigendum Wording

Insert after 9.5.1(2):

Within the body of a protected function (or a function declared immediately within a protected_body), the current instance of the enclosing protected unit is defined to be a constant (that is, its subcomponents may be read but not updated). Within the body of a protected procedure (or a procedure declared immediately within a protected_body), and within an entry_body, the current instance is defined to be a variable (updating is permitted).

the new paragraphs:

For a type declared by a protected_type_declaration or for the anonymous type of an object declared by a single_protected_declaration, the following language-defined type-related representation aspect may be specified:

Exclusive_Functions

The type of aspect Exclusive_Functions is Boolean. If not specified (including by inheritance), the aspect is False.

A value of True for this aspect indicates that protected functions behave in the same way as protected procedures with respect to mutual exclusion and queue servicing (see below).

A protected procedure or entry is an *exclusive* protected operation. A protected function of a protected type *P* is an exclusive protected operation if the Exclusive_Functions aspect of *P* is True.

Replace 9.5.1(4):

A new protected action is not started on a protected object while another protected action on the same protected object is underway, unless both actions are the result of a call on a protected function. This rule is expressible in terms of the execution resource associated with the protected object:

by:

A new protected action is not started on a protected object while another protected action on the same protected object is underway, unless both actions are the result of a call on a nonexclusive protected function. This rule is expressible in terms of the execution resource associated with the protected object:

Replace 9.5.1(5):

- *Starting* a protected action on a protected object corresponds to *acquiring* the execution resource associated with the protected object, either either for concurrent read-only access if the protected action is for a call on a protected function, or for exclusive read-write access otherwise;

by:

- *Starting* a protected action on a protected object corresponds to *acquiring* the execution resource associated with the protected object, either for exclusive read-write access if the protected action is for a call on an exclusive protected operation, or for concurrent read-only access otherwise;

Replace 9.5.1(7):

After performing an operation on a protected object other than a call on a protected function, but prior to completing the associated protected action, the entry queues (if any) of the protected object are serviced (see 9.5.3).

by:

After performing an exclusive protected operation on a protected object, but prior to completing the associated protected action, the entry queues (if any) of the protected object are serviced (see 9.5.3).

Replace 9.5.3(15):

- If after performing, as part of a protected action on the associated protected object, an operation on the object other than a call on a protected function, the entry is checked and found to be open.

by:

- If after performing, as part of a protected action on the associated protected object, an exclusive protected operation on the object, the entry is checked and found to be open.

Replace 9.5.3(23):

When the entry of a protected object is checked to see whether it is open, the implementation need not reevaluate the condition of the corresponding entry_barrier if no variable or attribute referenced by the condition (directly or indirectly) has been altered by the execution (or cancellation) of a protected procedure or entry call on the object since the condition was last evaluated.

by:

When the entry of a protected object is checked to see whether it is open, the implementation need not reevaluate the condition of the corresponding entry_barrier if no variable or attribute referenced by the condition (directly or indirectly) has been altered by the execution (or cancellation) of a call to an exclusive protected operation of the object since the condition was last evaluated.

Response

Add a boolean aspect "Exclusive_Functions" for protected types and single protected objects to forbid concurrent execution of protected functions.

Discussion

According to A(3/2), there is no correctness guarantee on concurrent calls involving overlapping objects passed by reference. Indeed, most implementations of the standard containers break when the same container is used concurrently, even when the container is declared constant.

The usual way of allowing concurrent access to unsafe objects is to encapsulate them in a protected object.

However, protected functions still allow concurrent access to (a constant view) of private objects. So the most natural way of writing a protected container is still unsafe:

```
protected type Protected_Container is
   procedure Insert (Key : in Key_Type; Element : in Element_Type);
   function Element (Key : in Key_Type) return Element_Type;
private
   Unsafe_Container : Ordered_Map_Instance.Map;
end Protected_Container;
```

Moreover, there is no easy way to actually realize this is unsafe, since it compiles without warning, and the problem may not show up in testing as the (current) implementation may not take advantage of the possibility of concurrent access, and even if it does, there is no guarantee that anything bad will happen (as it typical for race conditions).

Programmers aware about the issue can restrict themselves to use only protected procedures and entries, which guarantees non-concurrency on private data:

```
protected type Safe_Protected_Container is
   procedure Insert (Key : in Key_Type; Element : in Element_Type);
```

```
      procedure Element (Key : in Key_Type; Element : out Element_Type);
   private
      Unsafe_Container : Ordered_Map_Instance.Map;
   end Safe_Protected_Container;
```

However, such a procedure Element is counter-intuitive, and makes client code heavier because contained elements can no longer be used in a more complex expression, explicit temporary variables have to be used instead.

Also, this construction only works when Element_Type is definite. To use indefinite containers safely by using only protected procedures and entries, another layer is needed, e.g. using an indefinite holder or an access type.

Moreover, the whole workaround is fragile, since there is no way to ensure protected functions won't be carelessly added in the future by a programmer not aware of the rationale.

So the current version of Ada doesn't seem to adequately allow concurrency protection for objects whose read-only concurrent access is not safe, like standard containers, but it can be the case for user types as well.

It is much easier and foolproof to have an aspect for protected types, that make protected functions have concurrency semantics of protected procedures, ensuring non-concurrent use of private objects, while remaining syntactically a function.

From an implementation point of view, it would just mean that protected function calls would use the same semaphore as protected procedures; some implementations may even already use only one kind of semaphore for both. Implementation burden would likely range from null to minimal.

Locking behavior might be implemented directly in the body of a protected type. As such, we don't allow specifying the Exclusive_Functions aspect for derived protected types, so that the compiler doesn't need to generate a new body with different locking. That seems like an excessive implementation burden, especially as the intended usage would never change the aspect.

Similarly, we don't allow Exclusive_Functions on a protected interface. We don't want to have to come up with rules for inheritance and generic matching for this aspect (if there are multiple progenitors, they could have different values for the aspect); especially as the value of the aspect for an interface seems minimal (an interface has no bodies of its own).

The aspect works on a protected type rather than on individual protected operations. This avoids problems caused by internal calls between protected functions with different exclusivity. For instance:

```
   protected type Ugh is
      function Exists (Key : in Key_Type) return True
         with Exclusive_Function; -- Not Ada!!
      function Element (Key : in Key_Type) return Element_Type;
      ...
   end Ugh;

   protected body Ugh is
      function Element (Key : in Key_Type) return Element_Type is
      begin
         if Exists (Key) then
            ...
         end if;
      end Element;
      ...
   end Ugh;
```

In a case like this, the call to Exists would require exclusivity, but the outer call to Element would not. Thus, (depending on the implementation) there would be a need to acquire the exclusivity lock (or read-write lock) in the middle of a protected action, and a need to evaluate barriers at the end of the protected action even

though it started non-exclusive. Grabbing the lock could of course suspend the operation within a protected action, something that is not supposed to happen.

We can imagine adding additional rules to avoid problems like this, but that clearly is taking something simple and making it more complex.

The design principle for protected objects is that barriers do not have to notice changes to global data that happen outside of the protected actions of the PO; but that barriers do notice changes to (any) data made by the protected actions. Thus, it is necessary for protected functions with the Exclusive_Functions aspect to evaluate barriers after each call. Such a function can safely update global data that is under control of the protected object. (It's not safe to do so from a regular protected function, as it might be executed concurrently.)

An implementation can use the permission of 9.5.3(23) to avoid such re-evaluation of barriers if the barriers do not depend on any global data that might be modified by the function (this will often be the case).

It has been suggested that if barriers need to be re-evaluated, then the protected object should have read-write access within the function. That seems to be too large a change for an aspect.

It's always been the case that tying read-write to the the procedure syntax (and read-only to the function syntax) is unnecessarily limiting. All of the possible combinations make sense in some uses (that includes read-only access from a procedure, the least useful combination). Unfortunately, to fix that would require some sort of major syntactic change to protected types (perhaps explicitly specifying the protected object parameter so that the mode is exposed?) It is unlikely that we'll have enough interest in such a major change.

Note that it has always been the case that a protected function can modify global data (this includes objects allocated from a storage pool but only accessible from the protected object). Exclusive_Functions makes this a safer thing to do (it probably won't work today on an implementation that uses a separate read-only lock, although [as with any race condition] it may take a long time to fail in practice).

Thus, this is a useful feature in its own right; it's not exclusively tied to use for containers operations. Arguably, a more encompassing feature would be better, but it seems unlikely that a complex new feature would be worth the extra development time (both for the ARG and for implementers). The proposed feature is "good enough" to solve a variety of problems; it's unclear that a fancier feature would bring enough additional value to be worth the much higher costs.

Example

The example in the discussion could be written:

```
protected type Protected_Container
   with Exclusive_Functions is
   procedure Insert (Key : in Key_Type; Element : in Element_Type);
   function Element (Key : in Key_Type) return Element_Type;
private
   Unsafe_Container : Ordered_Map_Instance.Map;
end Protected_Container;
```

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0169**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **9.5.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0169 Pre- and Postconditions and requeues

Working Reference Number AI12-0090-1

Question

Interactions between pre/post-condition checking (for entries) and requeues appear to have some peculiar consequences.

Suppose entries E1 and E2 have pre- and post-conditions, a call to E1 requeues to E2, and E2 returns normally.

Is E2's precondition ever checked? (A requeue isn't a call!) Is E1's postcondition ever checked?

Is the answer to either of these questions a problem that needs to be fixed? (Yes.)

Summary of Response

The preconditions of a target entry are evaluated before a call is requeued. The postconditions of a target entry have to be the same as those on the current entry.

Corrigendum Wording

Insert after 9.5.4(5):

If the requeue target has parameters, then its (prefixed) profile shall be subtype conformant with the profile of the innermost enclosing callable construct.

the new paragraphs:

Given a requeue_statement where the innermost enclosing callable construct is for an entry *E1*, for every specific or class-wide postcondition expression *P1* that applies to *E1*, there shall exist a postcondition expression *P2* that applies to the requeue target *E2* such that

- *P1* is fully conformant with the expression produced by replacing each reference in *P2* to a formal parameter of *E2* with a reference to the corresponding formal paramter of *E1*; and
- if *P1* is enabled, then *P2* is also enabled.

The requeue target shall not have an applicable specific or class-wide postcondition which includes an Old attribute_reference.

If the requeue target is declared immediately within the task_definition of a named task type or the protected_definition of a named protected type, and if the requeue statement occurs within the body of that type, and if the requeue is an external requeue, then the requeue target shall not have a specific or class-wide postcondition which includes a name denoting either the current instance of that type or any entity declared within the declaration of that type.

Replace 9.5.4(7):

The execution of a requeue_statement proceeds by first evaluating the *procedure_or_entry_*name, including the prefix identifying the target task or protected object and the expression identifying the entry within an entry family, if any. The entry_body or accept_statement enclosing the requeue_statement is then completed, finalized, and left (see 7.6.1).

by:

The execution of a requeue_statement proceeds by first evaluating the *procedure_or_entry_*name, including the prefix identifying the target task or protected object and the expression identifying the entry within an entry family, if any. Precondition checks are then performed as for a call to the requeue target entry or subprogram. The entry_body or accept_statement enclosing the requeue_statement is then completed, finalized, and left (see 7.6.1).

Replace 9.5.4(12):

> If the requeue target named in the requeue_statement has formal parameters, then during the execution of the accept_statement or entry_body corresponding to the new entry, the formal parameters denote the same objects as did the corresponding formal parameters of the callable construct completed by the requeue. In any case, no parameters are specified in a requeue_statement; any parameter passing is implicit.

by:

> If the requeue target named in the requeue_statement has formal parameters, then during the execution of the accept_statement or entry_body corresponding to the new entry and during the checking of any preconditions of the new entry, the formal parameters denote the same objects as did the corresponding formal parameters of the callable construct completed by the requeue. In any case, no parameters are specified in a requeue_statement; any parameter passing is implicit.

Discussion

Let's assume that E1 and E2 are as defined in the question.

Entering any callable entity without evaluating its precondition is trouble, as the body may assume that the precondition is true. (This is assuming that the Assertion_Policy is Check, of course.) We need to ensure that E2's precondition holds before its body executes, whether or not a requeue is a call.

Thus, we require that a requeue check the enabled preconditions of the entry E2 before the call is requeued.

One expects the postcondition of E2 will be evaluated, but this is problematical if that postcondition includes 'Old references that were never evaluated on the original call. Moreover, the postcondition of E1 will never be evaluated. That means that any promises that the postcondition of E1 makes could very well be false.

Thus, we require that the postconditions of E1 and E2 be the same (fully conform). For external requeues, we further require that they don't contain any references to parts of the object (such as a task discriminant) which might have different values in the different task or protected objects. [The objects necessarily are different objects of the same type; otherwise the full conformance check would have failed (if the types are different), it's an immediate deadlock (if it's the same task object), or it's an immediate bounded error that probably raises Program_Error (if it's the same protected object). Thus the rule only needs to apply to types and not to single objects.] We allow such references if this is an internal call, as such references necessarily are to the same object and thus have the same value.

We do not attempt to prevent uses in postconditions of things like (instances of) Ada.Task_Attributes or Ada.Task_Identification.Current_Task even though those might yield different results depending on the identity of the task which evaluates the postcondition expression. That's because these can be used in any function called from the postcondition, including functions declared outside of the task. As such, we can't really prohibit such things, and doing a partial job doesn't seem very helpful. Additionally, Current_Task and the associated task attributes aren't even well-defined in a protected entry body, so they aren't useful in postconditions of protected entries. So we don't think the problematic uses will appear very often, and mixing them with requeues will be rarer still.

We considered the effect on Type Invariants. We eventually concluded that the primary use of type invariants is for the writer of the body, so a promise to the caller is less important. In addition, type invariants already have a number of other holes, so adding one more unlikely way to escape a check doesn't change the utility of the feature much. Whereas we believe the checking of postconditions cannot be evaded, thus we expect programmers to trust them and do not want to introduce any way to have enabled postconditions to be unchecked.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0170**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **9.7.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0170 Problematic examples for ATC

Working Reference Number AI12-0098-1

Question

9.7.4 contains two examples demonstrating how asynchronous transfer of control (usually abbreviated ATC) can be used. But neither of these examples can be assumed to work as described.

When the entry call or delay is completed, the abortable part is aborted. However, this doesn't mean the abortable part stops executing immediately. Ada only requires abortion at "abort completion points". (Annex D has additional requirements, but these examples are in the core language and should depend only on core requirements.)

In the first example, an input operation is not an abort completion point. Thus, the abortable part need not complete until some later time.

In the second example, if Horribly_Complicated_Recursive_Function contains only calculations, it won't have any abort completion points and therefore may not complete when the delay expires.

Unless the implementation is claiming Annex D support (and thus supporting D.6), neither of these examples need work as explained in the standard. Do we need some clarification here? (Yes.)

Summary of Response

The examples in 9.7.4 assume appropriate abort completion points.

Corrigendum Wording

Insert after 9.7.4(13):

```
select
   delay 5.0;
   Put_Line("Calculation does not converge");
then abort
   -- This calculation should finish in 5.0 seconds;
   --  if not, it is assumed to diverge.
   Horribly_Complicated_Recursive_Function(X, Y);
end select;
```

the new paragraph:

> Note that these examples presume that there are abort completion points within the execution of the abortable_part.

Response

The examples accurately reflect the intended use of the feature. However, the questioner is correct that neither of these examples are required to work usefully in the absence of D.6.

Therefore, we add a note that the examples assume abort completion points within the abortable part.

Discussion

9.8 says that a task only needs to be aborted at abort completion points. Of course, the implementation could support aborting it at other points, but that's not required of an Ada implementation (unless Annex D is supported, which is a separate issue).

Thus, we add an explanation after the examples that we're assuming that the abortable_parts include abort completion points.

For the second example, this is perfectly reasonable and possible -- the programmer must write their calculation so that it contains abort completion points, if they want to have a portable calculation.

For the first example, this comment is more problematical. Clearly, the user cannot control whether a language-defined subprogram contains any abort completion points. They can put one or more into Process_Command, but the I/O routines cannot be assumed to be abortable.

The question came up because the first example did not work on a Windows compiler. The initial response from an Ada language designer was that the compiler should be fixed. But Windows I/O operations are not interruptable in general (other threads can run, but the thread doing I/O has to wait for that I/O to complete); there are ways to make I/O operations interruptable but those have significant costs in complexity and overhead. Adding overhead to I/O in Windows programs just so ATC works are described in the examples would be silly and counterproductive.

(Aside: The problem occurs on Windows if a 1-thread implementation of ATC is used. One could use a 2-thread model to implement ATC on Windows to side-step these issues, but that would encur costs to create a thread for each ATC, as well as ensuring that exceptions are delivered properly. It's probable that such an implementation would allow ATC to "work", but it might be too slow to use in practice.)

We will not try to address this problem; if Get_Line doesn't include an abort completion point, the example probably won't work as intended. Hopefully, implementers will not be pressured into providing high-overhead I/O implementations just to make ATC examples like this one work.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0171**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **10.2.1; E.2.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0171 Variable state in pure packages

Working Reference Number AI12-0076-1

Question

In Ada 2012, the Rosen trick can be used to have variable state in a pure package. For instance:

```
package Not_So_Pure
   with Pure is
   type Outer;
   type Inner (Ref : access Outer) is limited null record;
   type Outer is limited record
      Self : Inner (Ref => Outer'access);
      Int : Integer := 123;
   end record;

   X : constant  Outer := (others => <>);
   -- clients can modify X.Self.Ref.all.Int
end;
```

Is this a problem? (Yes.)

Summary of Response

It is erroneous to change the value of a library-level constant object in a pure or remote-types package, except as part of its initialization or finalization.

Note that it is already erroneous to change the discriminants of an object within a construct where there is a "live" reference to a discriminant-dependent component of the object (3.7.2(4)).

Corrigendum Wording

Insert after 10.2.1(17):

A pragma Pure is used to specify that a library unit is *declared pure*, namely that the Pure aspect of the library unit is True; all compilation units of the library unit are declared pure. In addition, the limited view of any library package is declared pure. The declaration and body of a declared pure library unit, and all subunits that are elaborated as part of elaborating the library unit, shall be pure. All compilation units of a declared pure library unit shall depend semantically only on declared pure library_items. In addition to the places where Legality Rules normally apply (see 12.3), these rules also apply in the private part of an instance of a generic unit. Furthermore, the full view of any partial view declared in the visible part of a declared pure library unit that has any available stream attributes shall support external streaming (see 13.13.2).

the new paragraph:

Erroneous Execution

Execution is erroneous if some operation (other than the initialization or finalization of the object) modifies the value of a constant object declared at library-level in a pure package.

Insert after E.2.2(17):

- The Storage_Pool attribute is not defined for a remote access-to-class-wide type; the expected type for an allocator shall not be a remote access-to-class-wide type. A remote access-to-class-wide type shall not be an actual parameter for a generic formal access type. The Storage_Size attribute of a remote access-to-class-wide type yields 0; it is not allowed in an attribute_definition_clause.

the new paragraph:

Erroneous Execution

Execution is erroneous if some operation (other than the initialization or finalization of the object) modifies the value of a constant object declared in the visible part of a remote types package.

Response

(See !summary.)

Discussion

This is very important for distribution. The model of distribution is that pure packages can replicated in each partition. If there is variable state in such a package, it would require some means of synchronization, which was not an intended part of the distribution model.

This problem could not occur in Ada 95 pure packages as access types were not allowed. In Ada 2005, any modification of a constant via an access value was erroneous because of 13.9.1(13). However, 13.9.1(13) was modified to allow modifications to controlled and immutably limited constants, as these are commonly used operations which need to be implemented correctly by all compilers.

This modification of 13.9.1(13) opened up this problem; as such we replace the erroneousness in cases where it would actually be harmful.

It would be preferable to have a static rule rather than erroneous execution for such cases, but such a rule could not break privacy and thus would have to reject constants of private types (as they might have a controlled part). That would be unacceptably incompatible.

Another alternative would be to craft a Bounded Error rule which would require an exception to be raised if this occurs (dynamic rules can look through privacy). While this probably is better, it's unclear that it is worth the effort as erroneous execution was considered "good enough" for this case in Ada 95 and 2005.

We do not need to add wording to cover the update of discriminants of known-to-be-constrained objects, as such changes are already covered by the general Erroneousness rule in 3.7.2(4). This makes it erroneous to change the value of a discriminant of an object while there exist live references to discriminant-dependent components of the object.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0172**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **11.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0172 Raise exception with failing string function

Working Reference Number AI12-0062-1

Question

What happens when the string expression in a "raise Excep with String_Func(...)" propagates an exception? Which exception is raised? (The one propagated by String_Func).

Summary of Response

If the string expression in a raise_statement or raise_expression itself raises an exception, that exception is propagated (rather than the one named in the raise_statement or expression).

Corrigendum Wording

Insert after 11.3(4):

> To *raise an exception* is to raise a new occurrence of that exception, as explained in 11.4. For the execution of a raise_statement with an *exception*_name, the named exception is raised. If a *string*_expression is present, the expression is evaluated and its value is associated with the exception occurrence. For the execution of a re-raise statement, the exception occurrence that caused transfer of control to the innermost enclosing handler is raised again.

the new paragraph:

> NOTES
>
> 1 If the evaluation of a *string*_expression raises an exception, that exception is propagated rather than the one denoted by the *exception*_name of the raise_statement or raise_expression.

Response

11.3(4/2) says:

"If a string_expression is present, the expression is evaluated and its value is associated with the exception occurrence."

This sentence is marked as Redundant, but it's not completely redundant because it is the only text that talks about when the string_expression is evaluated. In any case, "Redundant" is not normative, so this wording controls the effect whether we intended it to or not.

When an expression is evaluated, any exceptions that it might cause are propagated (of course). The above sentence says that happens before the string_expression is associated with the exception occurrence. And that association has to happen before the exception occurrence created by raise statement (or expression) is propagated, because that occurrence isn't fully defined until the value of the message is "associated", and because an occurrence cannot meet the requirements on the exception message of 11.4.1(10.4/3) until the association has happened. In any case, propagating an occurrence that isn't fully initialized is nonsense.

Thus we conclude that the exception raised by string_expression has priority.

This is the preferred response in any case, as it means that a raise_statement and the equivalent call to Ada.Exceptions.Raise_Exception have precisely the same semantics.

Discussion

None needed.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0173**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **12.5.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0173 The actual for an untagged formal derived type cannot be tagged

Working Reference Number AI12-0036-1

Question

Consider:

```
generic
   type T1 is private;
   type T2 is new T1;
   type T3 is new T1;
package Pack1 is
   procedure Proc;
end Pack1;

package body Pack1 is

   V1 : T1;
   V2 : T2;
   V3 : T3;

   procedure Proc is
   begin
      V2 := T2(V1);  -- (A)
      V3 := T3(V2);  -- (B)
      V1 := T1(V2);  -- (C)
   end Proc;

end Pack1;

type Root is tagged null record;
type Child1 is new Root with record ... end record;
type Child2 is new Root with record ... end record;

package Inst is new Pack1 (T1 => Root, T2 => Child1, T3 => Child2);
```

Nothing in the Standard makes this illegal. But the conversions (A) and (B) clearly cannot be allowed, as they would leave the extension components uninitialized. Indeed, such conversions would be illegal if type T1 was a tagged private type (and the other types were type extensions). (Conversions like (C) are not a problem for this case.)

Some rule has to make this illegal, right? (Yes.)

Summary of Response

The actual for an untagged formal derived type cannot be a tagged type.

Corrigendum Wording

Replace 12.5.1(5.1):

> The actual type for a formal derived type shall be a descendant of the ancestor type and every progenitor of the formal type. If the formal type is nonlimited, the actual type shall be nonlimited. If the reserved word **synchronized** appears in the declaration of the formal derived type, the actual type shall be a synchronized tagged type.

by:

> The actual type for a formal derived type shall be a descendant of the ancestor type and every progenitor of the formal type. If the formal type is nonlimited, the actual type shall be nonlimited. The actual type for a formal derived type shall be tagged if and only if the formal derived type is a private extension. If the reserved word **synchronized** appears in the declaration of the formal derived type, the actual type shall be a synchronized tagged type.

Discussion

An alternative solution would be to adjust the type conversion rules to disallow converting the formal derived type to its ancestor type, if that type could be tagged (that is, is a formal private type or a formal derived type whose ancestor could be tagged).

We selected the simpler rule of disallowing the instantiation, as it seems less likely to cause compatibility problems. For instance, Ada doesn't have an "any scalar type" kind of generic formal; formal private types are often used for that purpose. In that case, conversions to the ancestor would be expected and likely, but the generic would not be intended to be instantiated with any composite type (like a tagged type). On the other hand, rejecting the instance is unlikely to be a problem, as most such cases will want to allow extension and thus will be using tagged private types as the ancestor. In addition, untagged formal derived types are rare, so compatibility issues are likely to be much rarer.

This solution is still incompatible, but only in cases where the formal derived type does not accurately describe the actual. As noted above, we believe such instances will be rare.

Note that the problem does not require the types to be declared in the same generic unit; the problem also appears in cases like:

```
generic
   type T1 is private;
package Pack1 is
   generic
      type T2 is new T1;
      type T3 is new T1;
   package Inner_Generic is ...
end Pack1;
```

or: generic type T1 is private; package Pack1 is ... end Pack1; generic type T2 is new T1; type T3 is new T1; package Pack1.Child is ... end Pack1.Child;

The matching rule works in these cases as well.

We make the rule symmetric as it is possible to match a formal private extension with a nontagged type:

```
generic
   type N is tagged private;
   type NC is new N with private;
package GG is
   ...
end GG;

package P is
   type T1 is private;
private
   type T1 is tagged ...
end P;

with P;
package Q is
   type T2 is new P.T1;
end Q;

with Q;
package body P is
   package G is new GG (T1, T2);
   ...
end P;
```

Here, T1 is tagged, but T2 is not tagged nor an extension. It's not clear that there is any semantic problem with allowing a case like this, but we're taking the safest option here.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0174**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0174 Private types and predicates

Working Reference Number AI12-0116-1

Question

The following appears to be legal:

```
package P is
   type Priv is private
      with Dynamic_Predicate => Is_OK (Priv);
   function Is_OK (P : Priv) return Boolean;
private
   function Is_Great (P : Priv) return Boolean;
   type Priv is null record
      with Dynamic_Predicate => Is_Great (Priv);
end P;
```

Both the private type and the full type are of course type declarations, and there doesn't seem to be anything preventing this. In particular, 13.1.1(14/3) only applies in a single aspect_specification, and these clearly are two different aspect_specifications. We *don't* have a rule preventing specifying the same aspect twice for the same entity (we do have such a rule for representation aspects and for operational aspects, but predicates are never clearly defined to be either of those kinds of aspects).

Should this be legal? (No.)

Summary of Response

An aspect cannot be specified on two views of the same entity.

Corrigendum Wording

Replace 13.1(9):

A representation item that directly specifies an aspect of a subtype or type shall appear after the type is completely defined (see 3.11.1), and before the subtype or type is frozen (see 13.14). If a representation item or aspect_specification is given that directly specifies an aspect of an entity, then it is illegal to give another representation item or aspect_specification that directly specifies the same aspect of the entity.

by:

A representation item that directly specifies an aspect of a subtype or type shall appear after the type is completely defined (see 3.11.1), and before the subtype or type is frozen (see 13.14).

Replace 13.1(9.1):

An operational item that directly specifies an aspect of an entity shall appear before the entity is frozen (see 13.14). If an operational item or aspect_specification is given that directly specifies an aspect of an entity, then it is illegal to give another operational item or aspect_specification that directly specifies the same aspect of the entity.

by:

An operational item that directly specifies an aspect of an entity shall appear before the entity is frozen (see 13.14).

If a representation item, operational item, or aspect_specification is given that directly specifies an aspect of an entity, then it is illegal to give another representation item, operational item, or aspect_specification that directly specifies the same aspect of the entity.

Discussion

Aspects are one per type, not one per view, so we need to prevent giving one on multiple views. For representation and operational aspects, this is done by the rules in 13.1. But a subtype predicate is neither of

those kinds of aspects (neither are preconditions, postconditions, categorization aspects, and others -- it's not the case that we can categorize all aspects this way).

The existing rules in 13.1(9/3) and 13.1(9.1/3) could be read to cover all aspects. That was purely unintentional; when these paragraphs were reworded to cover aspects as well as other kinds of representation/operational items, the rewording didn't restrict the kinds of aspects involved. Thus, the rewording can be read to cover all aspects. This makes 13.1(9/3) conflict with 13.1(9.1/3) - they both seem to apply in some cases.

We have decided to extend the rules in 13.1 to cover all aspects. The best way to do this was to drop the existing specific rules and rather create a more general rule that clearly applies to all kinds of items and specifications.

Arguably, such a rule belongs in 13.1.1 (as it is a rule applying to all aspect_specifications), but it also belongs in 13.1 (as it affects all of representation items, operational items, and aspect_specifications). Since it is already in 13.1, we have left it there, but added an AARM note to 13.1.1 so that language laywers will know that it exists.

An alternative would be to define both subtype predicates and type invariants as operational aspects. Since 13.1(9.1/3) allows operational aspects on any kind of entity, both could be treated as operational. However, that would drag in some inheritance rules that we probably don't want. In addition, we'd have to have a default value, and support confirming aspects. That might all have worked fine had it been done from the beginning, but now it seems like an invitation for future ARG work to correct introduced bugs.

We wouldn't have needed any additional rules for Type_Invariant'Class as that is not allowed on full definitions. We wouldn't have needed rules for any preconditions or postconditions as there already is a rule in 13.1.1 preventing use on bodies. Similarly, categorization aspects are required to be given in specifications. But the new rule is harmless in all of these cases.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0175**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0175 Representation of untagged derived types

Working Reference Number AI12-0109-1

Question

(1) There appears to be a hole related to 13.1(10/3).

The problem is that there is nothing forbidding specifying a type-related representation aspect of the *parent* type (Arr in the example below) after a derivation takes place (definition of Arrx), and that will create the same situation that 13.1(10/3) is intended to prevent.

For instance:

```
procedure rep_clause is
   package P is
      type Val is (A, B, C, D, E, F, G, H);
      type Arr is array (1 .. 16) of Val;
      procedure xxx (Arg : arr);
      type Arrx is new Arr;
      for Arr'Component_Size use 3;  --<< *parent* type aspect
      Input_Data  : Arrx;
   end p;

   package body p is
      procedure xxx (Arg : arr) is null;
   end p;
begin
   p.xxx (p.Input_Data);
end;
```

If Arr had been a by-reference type, it would have been impossible to do conversions between Arr and Arrx (as copying of by-reference types is not allowed during parameter passing).

Should this be fixed? (Yes.)

(2) A related question noticed during this is that the inheritance of aspect specifications by a derived type is not defined. Specifically, when do aspect specifications become inheritable, given that name resolution of aspect specifications is deferred in some cases? That is, if you derive from a parent type before it has been frozen, and the parent type had some aspect specifications, are they all inherited, even if they haven't even undergone name resolution?

This should be specified, right? (Yes.)

Summary of Response

Late representation changes of a parent type are banned if the parent type is a by-reference type.

13.1(15/3) is intended even if the aspect has not yet been resolved or evaluated.

Corrigendum Wording

Replace 13.1(10):

> For an untagged derived type, it is illegal to specify a type-related representation aspect if the parent type is a by-reference type, or has any user-defined primitive subprograms.

by:

> For an untagged derived type, it is illegal to specify a type-related representation aspect if the parent type is a by-reference type, or has any user-defined primitive subprograms. Similarly, it is illegal to specify a nonconfirming type-related representation aspect for an untagged by-reference type after one or more types have been derived from it.

Discussion

Considering the questions separately:

(1)

The rule 13.1(10/3) is a strange combination of a required semantic rule and a methodological restriction. It is intended to prevent the need to make implicit conversions when calling an inherited primitive subprogram.

In some cases (such as limited by-reference types), making a copy of an object is prohibited and therefore it is impossible to make any implicit conversion. For such cases, a rule like 13.1(10/3) is absolutely required.

For other cases (such as fixed point types with different values of 'Small), doing an implicit conversion could be done, but it would have an undesirable semantic effect (potential loss of precision, especially when the conversions are made both ways as for in out parameters). Here, a rule like 13.1(10/3) is definitely desirable.

Finally, there are cases where there is no semantic problem as the conversions are semantics-preserving. These mainly are cases where the types have different but equivalent representations, such as packed and unpacked versions of an untagged record, or arrays with different component sizes. Here, 13.1(10/3) provides a purely methodological restriction, preventing potentially expensive implicit conversions (but also preventing inexpensive implicit conversions).

Methodological restrictions are always questionable, as they merely prevent a programmer from doing something that they may actually need to do. After all, there are plenty of operations that can be quite expensive in practice (controlled types, equality of composites, and so on). Generally, it's better to let knowledgable users write the code that they need (especially given that 13.6 recommends this approach for representation conversions), and give warnings if that leads to overly expensive code.

13.1(10/3) is intended to require explicitly converting between two different representations. That seems admirable, until one realizes that the effect is to require the elimination of all primitive operations of a (record) type in order to use Ada-language conversions. That means that the standard organization of putting a type and its operations together in a package cannot be followed - the operations have to placed somewhere else in order to take advantage of compiler-generated representation change. For many types (especially ones in packages already in use for which moving operations would require changes to many clients), the net effect is that derivation is infeasible and the representation changes end up getting written manually by the user, with all of the potential for mistakes and higher execution costs that entails.

The question shows that users have been clever in finding ways around this restriction. Moreover, there is a concern that fixing this hole would have significant compatibility effects, because users have been using this loophole over the years; it is claimed that GNAT has implemented this "feature" for 20 years.

As such, we only fix as much of the loophole as necessary to eliminate semantic problems. We extend the rules from AI05-0246-1 so that they cover such late representation changes. This is the minimum that we have to do.

It would be appealing to relax 13.1(10/3) to match the new rule. After all, this provides a way to end-run 13.1(10/3), much like generics provided a way to end-run the ban on redefinition of "=" in Ada 83. However, that's going beyond the question of the AI, and some misguided people think that the methodological restriction is a good idea.

An alternative fix would be to ban the derivation of an untagged type with primitives in the same package. That would certainly fix the problem, but it was judged to be too incompatible.

Another alternative fix would be to say that the representation of the untagged derived type with primitives is the same as the parent type if that derivation is in the same package as the derived type. For most implementations, that would just fix a bug. But it would be inconsistent if in fact the implicit conversions were supported, as is claimed for GNAT. Thus it is rejected.

Note that this is a nasty check. It requires rejecting something based on how it is used. Typical implementations do not keep track of how things are used, so at best implementing this check will require building a data structure solely for the purpose of this check (with all the complications of a little used data

structure). At worst, it would require a brute-force search of the package specification for every type-related aspect specification. (That could be a significant time hit if the package is large and most types have representation clauses -- this describes Claw exactly). And it's unfortunate that all of this complication happens for something that is quite unlikely.

(2)

13.1(15/3) says:

A derived type inherits each type-related representation aspect of its parent type that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent type from the grandparent type.

It seems clear enough that the aspect is inherited in such a case. It would have to be inherited without knowing the actual value. At the point of the derivation, the only thing that could be inherited is the unresolved and unevaluated expression. An alternative view would have us somehow inheriting the future value of the aspect. That would require a substantial implementation cost used solely for this rather unlikely case (derivations after freezing of the parent type -- including in other packages -- would not use it, the current implementation [whatever it is] sufficies in that case). As such, we assume the simpler model (which would just reuse the existing aspect mechanism) and see where it leads.

There is clearly a possibility of anomalies given that the aspect has neither been resolved or evaluated at the point of the derived type (presuming that no freezing point of the parent type has occurred).

First of all, inheriting the raw aspect means that the aspect would be evaluated twice. This seems like it would be a problem, but it isn't as most type-related aspects require static expressions. Those that don't (like Storage_Size and Storage_Pool) are not allowed on derived types. Thus we conclude that there is no problem from re-evaluation of aspects.

Second, the question arises is to whether the two copies could ultimately get different values. We certainly would not want that, as the entire issue with 13.1(10/3) is to ensure the values are the same. But 13.1.1(13/3) prevents that. Anything that would make the values different would also make one or the other expressions resolve differently -- and that is illegal.

Third, one wonders if 13.1.1(13/3) could make the inherited aspect illegal when the original aspect was legal. But that can't happen because the redefinition of an entity would have to follow the derived type in order that the inherited aspect is illegal. The first freezing point of the derived type would have to be before the redefined entity. But that point must either freeze the parent type (in which case it is also illegal) or the parent type would have to have been frozen previously (again making it illegal). It is possible for the parent type's aspect to be illegal and the derived type to be legal, but that isn't related to the derivation so it is irrelevant for this purpose.

Thus, we conclude that the model of inheriting the raw (unresolved and unevaluated) aspect works. As such, we simply add an AARM note to verify that this is the intent and claim victory. :-)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0176**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0176 Pre and Post are not allowed on any subprogram completion

Working Reference Number AI12-0105-1

Question

13.1.1(18/3) says "A language-defined aspect shall not be specified in an aspect_specification given on a subprogram_body or subprogram_body_stub that is a completion of another declaration."

This is intended to prevent hiding aspects like Pre and Post on a body, as they are intended to be known to callers.

However, an expression_function that is a completion is not a subprogram_body, so the above doesn't apply to it. Similarly for a null_procedure that is a completion. The rule should apply those as well, right? (Yes.)

Summary of Response

No language-defined aspect (such as Pre or Post) is allowed on any subprogram completion.

Corrigendum Wording

Replace 13.1.1(18):

> A language-defined aspect shall not be specified in an aspect_specification given on a subprogram_body or subprogram_body_stub that is a completion of another declaration.

by:

> A language-defined aspect shall not be specified in an aspect_specification given on a completion of a subprogram or generic subprogram.

Discussion

While an expression_function that is a completion is not a subprogram_body (syntax font), it is a body (regular font) and it surely is a subprogram (regular font). So we redo the wording to use semantic terms. The alternative of listing each of the kinds of declaration would get wordy and would potentially pose a future maintenance hazard (should any additional kinds of body get defined).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0177**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.1.1; 13.13.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0177 Write'Class aspect

Working Reference Number AI12-0106-1

Question

ACATS 4.0 test BDD2005 contains a number of cases like:

```
type My_Tagged_4 is abstract tagged null record
   with Write'Class => Good_Write3;-- OK.
```

The test assumes this is the notation for specifying class-wide stream aspects.

However, an argument can be made that these aspects cannot be specified with an aspect_specification since they need to be specified on the type T'Class, and that type never has an explicit declaration.

Another argument can be made that they can be specified, but their name is "Class'Write". That can be refuted by the fact that there is no such syntax defined for aspect_specifications.

So, do we want to be able to specify Write'Class as suggested by the new ACATS test? (Yes.)

Summary of Response

Class-wide stream attributes can be specified with the syntax <aspect>'Class, but they are never inherited.

Corrigendum Wording

Replace 13.1.1(28):

If the aspect_mark includes 'Class, then:

by:

If the aspect_mark includes 'Class (a *class-wide aspect*), then, unless specified otherwise for a particular class-wide aspect:

Replace 13.13.2(38):

The stream-oriented attributes may be specified for any type via an attribute_definition_clause. The subprogram name given in such a clause shall statically denote a subprogram that is not an abstract subprogram. Furthermore, if a stream-oriented attribute is specified for an interface type by an attribute_definition_clause, the subprogram name given in the clause shall statically denote a null procedure.

by:

The stream-oriented attributes may be specified for any type via an attribute_definition_clause. Alternatively, each of the specific stream-oriented attributes may be specified using an aspect_specification on any type_declaration, with the aspect name being the corresponding attribute name. Each of the class-wide stream-oriented attributes may be specified using an aspect_specification for a tagged type *T* using the name of the stream-oriented attribute followed by 'Class; such class-wide aspects do not apply to other descendants of *T*.

The subprogram name given in such an attribute_definition_clause or aspect_specification shall statically denote a subprogram that is not an abstract subprogram. Furthermore, if a specific stream-oriented attribute is specified for an interface type, the subprogram name given in the attribute_definition_clause or aspect_specification shall statically denote a null procedure.

Discussion

We want to be able to use aspect_specifications in as many cases as possible, so we want some way to specify the class-wide stream aspects. Moreover, both the syntax of aspect_specifications and consistency argue that the form of such names is <aspect>'Class. Finally, aspect specifications are associated with explicit

declarations only, and T'Class is declared implicitly. So we propose that the attribute XYZ of T'Class can be specified by specifying the XYZ'Class aspect of T.

Of course, we want this for all four stream attributes.

Note that 13.1.1(29/3) says that a class-wide aspect of a type applies to all descendants of the type. We don't want that to happen in this case (as it doesn't happen for an attribute_definition_clause, and we surely don't want these to act different), so we adjust the wording in 13.1.1(28/3) to say "unless specified otherwise..." and then in 13.13.2 indicate that it doesn't apply to the class-wide stream aspects.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0178**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0178 Aspects of library units

Working Reference Number AI12-0154-1

Question

(1) The first sentence 13.14(3/4) determines when the contents of a library package are frozen. But there does not seem to be any definition of when the library package itself is frozen. Taken literally, that would imply that it is frozen at the end of package Standard (as all library units are nested within it, and its end would trigger 13.14(3/4)). But is there in fact an end to Standard, given that a subsequent compilation can add additional units at any time?

This matters because 13.1.1(37/3) and 13.14(7.2/3) say that expressions in an aspect specification are evaluated no sooner than the first freezing point (and are resolved based on the end of the enclosing declaration list [13.1.1(11/3)]). We obviously need to know how aspects of library packages are resolved and evaluated.

When is a library package frozen?

(2) Most library package aspects correspond to library unit pragmas. Library unit pragmas take effect immediately. If the aspect is evaluated at the end of the package or even later, then the aspect is quite different than the associated pragma.

For instance:

```
package P with Pure => Purity is
   -- Var : Integer;
   Purity : constant Boolean := True;
end P;
```

If this is legal, then compilers need to make a retroactive check for Purity on declarations. This is a new capability not contemplated when these pragmas were given aspect forms, and it does not seem particularly useful.

When does aspect Pure and similar aspects take effect? (Immediately.)

Summary of Response

The expression of an aspect associated with library unit pragma is resolved and evaluated immediately.

Corrigendum Wording

Replace 13.1.1(32):

> Any aspect specified by a representation pragma or library unit pragma that has a local_name as its single argument may be specified by an aspect_specification, with the entity being the local_name. The aspect_definition is expected to be of type Boolean. The expression shall be static.

by:

> Any aspect specified by a representation pragma or library unit pragma that has a local_name as its single argument may be specified by an aspect_specification, with the entity being the local_name. The aspect_definition is expected to be of type Boolean. The expression shall be static. Notwithstanding what this International Standard says elsewhere, the expression of an aspect that can be specified by a library unit pragma is resolved and evaluated at the point where it occurs in the aspect_specification, rather than the first freezing point of the associated package.

Discussion

There are a number of answers to question (1), and which ones make the most sense depends upon the usage of any aspects or representation pragmas that apply to a package. It's easy to imagine package aspects that need visibility into the package. For instance:

```
package P with Finalize => Cleanup is
   ...
   procedure Cleanup;
end P;
```

or

```
package P with Package_Invariant => All_Is_Copacetic is
   ...
   function All_Is_Copacetic return Boolean;
end P;
```

We also likely want to allow the use of trailing pragmas to specify values.

Thus, we probably want to freeze the package no sooner than the end of the enclosing compilation.

However (question (2)), it's clear that we want language-defined aspects that are associated with library unit pragmas to be resolved and frozen immediately. There is no benefit to allowing examples where retroactive checking of categorization restrictions is required; it seems that such examples would only appear in ACATS-style tests. Moreover, there was no intention of requiring additional work for implementers beyond that needed to directly support the aspect syntax. Having the effect of such aspects start immediately requires immediate evaluation of the expressions given for such aspects.

It turns out that the only language-defined aspects of a package are associated with library unit pragmas. Since we don't want to prevent implementation-defined aspects like the above, we do not want to freeze packages early. As such, we'll need a rule not related to freezing to handle package aspects. Once that is the case, we no longer need to answer the freezing question, so we don't. (Although we may need to do so in the future should we add any language-defined package aspects that need to be evaluated at the end of a library package, rather than the beginning.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0179**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.2; C.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0179 Independence and Representation clauses for atomic objects

Working Reference Number AI12-0001-1

Question

The Recommended Level of Support implies that it is required to support pragma Pack on types that have Atomic_Components, even to the bit level. Is this the intent? (No.)

Summary of Response

[Editor's note: This AI was carried over from Ada 2005.]

Pack doesn't require tight packing in infeasible cases (atomic, aliased, by-reference types, independent addressability).

Corrigendum Wording

Delete 13.2(6.1):

If a packed type has a component that is not of a by-reference type and has no aliased part, then such a component need not be aligned according to the Alignment of its subtype; in particular it need not be allocated on a storage element boundary.

Insert after 13.2(7):

The recommended level of support for pragma Pack is:

the new paragraph:

- Any component of a packed type that is of a by-reference type, that is specified as independently addressable, or that contains an aliased part, shall be aligned according to the alignment of its subtype.

Replace 13.2(8):

- For a packed record type, the components should be packed as tightly as possible subject to, the Sizes of the component subtypes, and subject to any record_representation_clause that applies to the type; the implementation may, but need not, reorder components or cross aligned word boundaries to improve the packing. A component whose Size is greater than the word size may be allocated an integral number of words.

by:

- For a packed record type, the components should be packed as tightly as possible subject to the above alignment requirements, the Sizes of the component subtypes, and any record_representation_clause that applies to the type; the implementation may, but need not, reorder components or cross aligned word boundaries to improve the packing. A component whose Size is greater than the word size may be allocated an integral number of words.

Replace 13.2(9):

- For a packed array type, if the Size of the component subtype is less than or equal to the word size, Component_Size should be less than or equal to the Size of the component subtype, rounded up to the nearest factor of the word size.

by:

- For a packed array type, if the Size of the component subtype is less than or equal to the word size, Component_Size should be less than or equal to the Size of the component subtype, rounded up to the nearest factor of the word size, unless this would violate the above alignment requirements.

Replace C.6(8.1):

When True, the aspects Independent and Independent_Components *specify as independently addressable* the named object or component(s), or in the case of a type, all objects or components of that type. All atomic objects are considered to be specified as independently addressable.

by:

When True, the aspects Independent and Independent_Components *specify as independently addressable* the named object or component(s), or in the case of a type, all objects or components of that type. All atomic objects and aliased objects are considered to be specified as independently addressable.

Replace C.6(10):

It is illegal to specify either of the aspects Atomic or Atomic_Components to have the value True for an object or type if the implementation cannot support the indivisible reads and updates required by the aspect (see below).

by:

It is illegal to specify either of the aspects Atomic or Atomic_Components to have the value True for an object or type if the implementation cannot support the indivisible and independent reads and updates required by the aspect (see below).

Replace C.6(11):

It is illegal to specify the Size attribute of an atomic object, the Component_Size attribute for an array type with atomic components, or the layout attributes of an atomic component, in a way that prevents the implementation from performing the required indivisible reads and updates.

by:

It is illegal to specify the Size attribute of an atomic object, the Component_Size attribute for an array type with atomic components, or the layout attributes of an atomic component, in a way that prevents the implementation from performing the required indivisible and independent reads and updates.

Delete C.6(21):

If a pragma Pack applies to a type any of whose subcomponents are atomic, the implementation shall not pack the atomic subcomponents more tightly than that for which it can support indivisible reads and updates.

Insert after C.6(24):

NOTES:

9 An imported volatile or atomic constant behaves as a constant (i.e. read-only) with respect to other parts of the Ada program, but can still be modified by an "external source."

the new paragraph:

10 Specifying the Pack aspect cannot override the effect of specifying an Atomic or Atomic_Components aspect.

Response

Resolve the difference by eliminating C.6 (21) and changing 13.2 (6.1/2) to be a recommended level of support where by-reference, aliased, and atomic objects must be aligned according to subtype.

Change 13.2(8 and 9) to add an exception for components that have alignment requirements as detailed above.

In C.6(8.1/3), add "and aliased objects" after "atomic objects".

In C.6(10-11), add "and Independent" after indivisible.

Delete C.6 (21) as it is no longer required.

Discussion

The idea of Pack is that if it's infeasible to pack a given component tightly (because it is atomic, aliased, of a by-reference type, or has independent addressability), then Pack is not illegal; it just doesn't pack as tightly as it might without the atomic, volatile, etc.

This was always the intent, but the Recommended Level of Support (RLS) contradicted it.

By making the alignment requirement part of the Recommended Level of Support eliminates the conflict between the RLS and the intent.

Note that we require that aliased objects are always independently-addressable. We want dereferences to always be task-safe in this way; modifying an object through a dereference will never clobber some adjacent component (even momentarily).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0180**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0180 Has_Same_Storage on objects of size zero

Working Reference Number AI12-0077-1

Question

The description of Has_Same_Storage should be explicit about the result of X'Has_Same_Storage(Y) when X'Size = Y'Size = 0. The current text could be read either way. (By contrast, X'Overlaps_Storage(Y) is clearly false because the objects cannot "share at least one bit".)

What is the result? (It is False.)

Summary of Response

X'Has_Same_Storage(Y) returns False if X or Y or both occupy no bits.

Corrigendum Wording

Replace 13.3(73.4):

> The actual parameter shall be a name that denotes an object. The object denoted by the actual parameter can be of any type. This function evaluates the names of the objects involved and returns True if the representation of the object denoted by the actual parameter occupies exactly the same bits as the representation of the object denoted by X; otherwise, it returns False.

by:

> The actual parameter shall be a name that denotes an object. The object denoted by the actual parameter can be of any type. This function evaluates the names of the objects involved. It returns True if the representation of the object denoted by the actual parameter occupies exactly the same bits as the representation of the object denoted by X and the objects occupy at least one bit; otherwise, it returns False.

Response

(See !summary.)

Discussion

The intended use case of Has_Same_Storage and Overlaps_Storage is to answer the question: "if I write to A, is there a risk that I'm ruining B?". Since this cannot happen for zero-sized objects, the answer should be False if either operand is zero-sized. As the questioner notes, this is clear for Overlaps_Storage, but we need wording for Has_Same_Storage.

We also want to preserve the relationship that Has_Same_Storage imples Overlaps_Storage. If Overlaps_Storage is False, this means that Has_Same_Storage also has to be False.

This interpretation might cause some runtime overhead, but only in the case where the size of both operands are determined at runtime and both can be zero. In this case, Has_Same_Storage will need a size check anyway (so the cost of reading/calculating the sizes will be included no matter what rule we decide on), so the extra cost of a comparison against zero will not be particularly significant.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0181**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **13.11**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0181 Details of the storage pool used when Storage_Size is specified

Working Reference Number AI12-0043-1

Question

The ACAA received a test dispute that claims that language in 13.11(18) allows pretty much anything the implementor wants, as none of the properties of the implementation-defined pool are specified, other than that the memory is reclaimed when the scope is left.

The intent was that implementations would continue to implement this as it was defined in Ada 83, with the difference that reasonable rounding up is allowed, but this surely shouldn't be left to the imagination.

Should this wording be tightened up? (Yes.)

Summary of Response

The storage pool used by an access type that has Storage_Size specified must not allocate additional storage when the initial amount is exhausted, and the memory must not be shared with other types unless that is explicitly requested by attaching the pool to another type.

Corrigendum Wording

Replace 13.11(18):

> If Storage_Size is specified for an access type, then the Storage_Size of this pool is at least that requested, and the storage for the pool is reclaimed when the master containing the declaration of the access type is left. If the implementation cannot satisfy the request, Storage_Error is raised at the point of the attribute_definition_clause. If neither Storage_Pool nor Storage_Size are specified, then the meaning of Storage_Size is implementation defined.

by:

> If Storage_Size is specified for an access type *T*, an implementation-defined pool *P* is used for the type. The Storage_Size of *P* is at least that requested, and the storage for *P* is reclaimed when the master containing the declaration of the access type is left. If the implementation cannot satisfy the request, Storage_Error is raised at the freezing point of type *T*. The storage pool *P* is used only for allocators returning type *T* or other access types specified to use *T*'Storage_Pool. Storage_Error is raised by an allocator returning such a type if the storage space of *P* is exhausted (additional memory is not allocated).
>
> If neither Storage_Pool nor Storage_Size are specified, then the meaning of Storage_Size is implementation defined.

Discussion

We could also mention that a call to Deallocate (via an instance of Unchecked_Deallocation) for such may, but is not required to, return the memory to the pool for further use. We didn't mention that because it seems like overspecification; we mainly care that the expected number of allocations (with reasonable rounding) work and then Storage_Error is raised.

Note that while the definition of Storage_Pools talks about what the operations are "intended" to do, this wording exists only because a user-defined pool can do anything it wants to, whether or not that makes any sense compared to the intent. For implementation-defined pools, it is expected that they follow the intent - there is no reason for them to deviate. We could try to add some normative wording to that effect, but it's hard to word and it only seems necessary to someone who is trying to cheat.

We changed the point where Storage_Error is raised in this wording. The existing wording says that it is raised at the point of the attribute_definition_clause. If Storage_Size is specified with an aspect (likely in new code) -- then no attribute_definition_clause is available to be the point of raising the error. In addition, the amount to round up might not be available until the type is frozen (if it depends on the designated type). In

any case, the exception will be raised in the same declarative part, and it would take a truly pathological program to be able to tell the difference (having a declaration with a side-effect occuring between the attribute_definition_clause and the freezing point).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0182**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.11.2; 13.11.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0182 Dangling references

Working Reference Number AI12-0148-1

Question

(1) 13.11.2(10/2) says that the object deallocated by an instance of Unchecked_Deallocation ceases to exist after the call to that instance. There is no such wording in 13.11.5. That implies that Deallocate_Subpool cannot free the memory used by the objects in the subpool, as they still exist and thus can be accessed without causing erroneous execution.

But that is madness, since it defeats the entire purpose of having subpools. Should we add wording about the objects ceasing to exist? (Yes.)

(2) 13.11.2(9/3) says that Deallocate is called during the execution of an instance of Unchecked_Deallocation. 13.11.2(10/2) says that the object deallocated by an instance of Unchecked_Deallocation ceases to exist after the call to that instance. 13.11(21) says that execution is erroneous if the allocated storage from a pool is used for any other purpose while the pool element still exists. That means that there is a (short) window after a call to Deallocate when the object contained still exists, which means that actually making memory available for reuse immediately within Deallocate is an incorrect implementation (as it is possible that the memory would be used for another purpose before the object ceases to exist). But that is madness, since it means that the typical way that user-defined pools are constructed is wrong.

We should fix the wording so the deallocated object ceases to exist before Deallocate is called, right? (Yes.)

(3) Consider what happens if a pool object and the access type that uses it are in the same scope:

```
A_Pool : Some_Pool_Type;

type Acc is access ...
   with Storage_Pool => A_Pool;
```

Considering only the objects allocated for Acc and not explicitly deallocated, 7.6.1 says that the following happens in the listed order:

> The collection of type Acc is finalized. A_Pool is finalized. The collection and A_Pool cease to exist when the master is left.

If the finalization of A_Pool frees the memory associated with the collection (which is a very likely implementation), then it will have been freed before the objects in the collection have ceased to exist. Thus, if the memory is reused immediately, then the program execution is erroneous. But that is madness, since it means that the typical way that user-defined pools are constructed is wrong.

Should this be changed? (No.)

(4) Consider:

> Ptr1 := new Designated; Ptr2 := Ptr1; Free (Ptr1); Ptr3 := new Designated; if Ptr2 = Ptr3 then -- !

If the implementation uses just a machine address for the representation of an access value, and if Free makes the memory used by the allocator for Ptr1 available for reuse, and the allocator for Ptr3 in fact uses the same memory, then Ptr2 and Ptr3 will in fact contain exactly the same bit pattern. However, 4.5.2(12) makes it pretty clear that the only time that the equality of Ptr2 and Ptr3 can return True is when the designated objects are the same. That's clearly not the case here (the first allocated object no longer exists, but if we ignore that it's not the same as the second allocated object, and if we take that into account, something that doesn't exist still can't be the same as something that does exist). Thus, the implementation has to avoid reusing memory, or has to use some more complex represenation for access values. But that is madness, since it means that typical Ada implementations (in particular ones that match the representation of an access value with the one used by C) are wrong (and because of an obscure corner-case at that).

Should this be fixed? (Yes.)

Summary of Response

The objects in a subpool cease to exist during a call to Unchecked_Deallocate_Subpool before the call to Deallocate_Subpool.

The object deallocated by an instance of Unchecked_Deallocation ceases to exist before the call to Deallocate.

It is a Bounded Error to evaluate the value of an access object that is a dangling reference; execution is erronous if an access object whose value is a dangling reference is dereferenced.

Corrigendum Wording

Replace 13.11.2(10):

> After Free(X), the object designated by X, and any subcomponents (and coextensions) thereof, no longer exist; their storage can be reused for other purposes.

by:

> After the finalization step of Free(X), the object designated by X, and any subcomponents (and coextensions) thereof, no longer exist; their storage can be reused for other purposes.

Insert after 13.11.2(15):

> In the first two cases, the storage for the discriminants (and for any enclosing object if it is designated by an access discriminant of the task) is not reclaimed prior to task termination.

the new paragraphs:

> An access value that designates a nonexistent object is called a *dangling reference*.
>
> If a dangling reference is dereferenced (implicitly or explicitly), execution is erroneous (see below). If there is no explicit or implicit dereference, then it is a bounded error to evaluate an expression whose result is a dangling reference. If the error is detected, either Constraint_Error or Program_Error is raised. Otherwise, execution proceeds normally, but with the possibility that the access value designates some other existing object.

Insert after 13.11.5(7):

- Any of the objects allocated from the subpool that still exist are finalized in an arbitrary order;

the new paragraph:

- All of the objects allocated from the subpool cease to exist;

Discussion

We define the term "dangling reference" to characterize the case where an access value designates a nonexistent object. We then define the results of various operations on such values. The definition partially subsumes the rule of 13.11.2(16/3) about evaluating the name of a nonexistent object, but we leave that rule as we cannot be certain that all such cases are covered. This fixes question (4).

We also add wording to 13.11.5 to say that the objects in the subpool cease to exist after finalization, but before the call to Deallocate_Subpool. That fixes question (1).

We move the wording in 13.11.2 to says that the object being deallocated ceases to exist after finalization but before the call to Deallocate. That fixes question (2).

We do not try to fix (3). It was suggested that we say that the collection ceases to exist when the pool object is finalized. However, all that does is move the bump under the rug, as any use of the objects in the collection after finalization of the storage pool would then become erroneous. Such access is currently not erroneous by itself (although the implementation of the pool most likely would make it erroneous anyway).

```
with the current rules, a pool implemented as expected could make execution of
the program formally erroneous if the memory is reused before the objects
```

cease to exist. However, a program may be formally erroneous, but no bad effects will occur unless the program tried to **use** the (already finalized) objects **while** they still exist after the pool was finalized (**and** thus the objects **do not** have any memory assigned to them). That will **not** affect the vast majority **of** programs as no one will **use** those objects after finalization. Thus the problem **is** more one **of** formal definition than one **of** any practical importance. As 7.6.1 **is** a very complex set **of** wording, rewriting it to fix this problem **is not** appealing. Therefore, we chose to **not** fix the problem **at** this time.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0183**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.11.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0183 Language-defined packages and aspect Default_Storage_Pool

Working Reference Number AI12-0136-1

Question

13.11.3(5/3) defines the aspect Default_Storage_Pool that can be used on an instance to specify (or override) the default pool used in that instance.

We're wondering about the effect of that aspect on language-defined generic units. For instance, it would be useful in some circumstances to use some user-defined pool to do allocations in one of the containers, such as Ada.Containers.Vectors. Is this expected to work? (Maybe.) What happens if the pool does something that the instance isn't expecting, like deallocating memory early? (Whatever 13.11 says.)

Summary of Response

The effect of the aspect Default_Storage_Pool on language-defined generic units is implementation-defined.

Corrigendum Wording

Insert after 13.11.3(5):

> The language-defined aspect Default_Storage_Pool may be specified for a generic instance; it defines the default pool for access types within an instance. The expected type for the Default_Storage_Pool aspect is Root_Storage_Pool'Class. The aspect_definition must be a name that denotes a variable. This aspect overrides any Default_Storage_Pool pragma that might apply to the generic unit; if the aspect is not specified, the default pool of the instance is that defined for the generic unit.

the new paragraph:

> The effect of specifying the aspect Default_Storage_Pool on an instance of a language-defined generic unit is implementation-defined.

Discussion

Changing the default pool could cause issues for a language-defined package that was not expecting such a change. If the pool didn't support some feature that the language-defined package was expecting, the package might fail to operate as expected.

Similarly, many pools allow deallocation of memory via techniques other than the standard language-defined ones; deallocating memory that a language-defined package is still using would almost certainly crash the partition.

13.11(21) declares a malfunctioning Allocate routine erroneous. The wording of that rule covers premature or malfunctioning Deallocate routines, via the statement about the memory not being used for any other purpose. We believe this wording is general enough that it would also apply within a language-defined unit, even though the Allocate calls are not visible in the specification.

There doesn't seem to be a counterpart to 13.11(21) for pools that support subpools. This is an issue that's not directly related to this question, so it is handled in a separate AI.

Currently, there is no requirement on how language-defined units are implemented. As such, they might use no allocators at all, or use a custom pool (for group deallocations, perhaps), or use the default pool. This would be portability problem if users start expecting specifying the pool on an instance of a language-defined package to have a particular effect.

We need to have some rules that apply to the use of aspect Default_Storage_Pool on a language-defined generic, so that users know what is and is not expected to work. We adopted the easiest rule, which is just to say that it is implementation-defined. This puts no requirements on implementers other than to document whether the default storage pool is used by a language-defined generic.

Note that the use of the default pool by language defined generics varies from never (Generic_Elementary_Functions) to unlikely (Bounded containers) to maybe (Discrete_Random), to likely (Direct_IO, for file management) to certain (Indefinite containers). It's unclear whether we would want to have the same rules for all of these packages.

For instance, for Pure packages, there should be no state and thus no allocators. For such packages, there would be no effect to any rule.

If we wanted to make it work for some packages, we could have used a rule like:

Implementation Requirements

If the language-defined generic unit <whatever> declares an access type which is used to allocate memory, it shall use the default pool.

AARM Reason: This allows using aspect Default_Storage_Pool to force allocations to use a specified pool. Note that we don't make any requirements on the size or number of allocations from the default pool, so any pool specified as the Default_Storage_Pool for an instance of a language-defined package ought to support any reasonable allocations.

Even wording like this isn't going to allow anything useful for pools supporting subpools; that would require adding subpool parameters somewhere in the definition of the package -- at that point we've lost all transparency.

Also note that en-mass deallocations only could be safe after the instance has been destroyed (gone out of scope); we would not want to put any additional requirements on how language-defined packages manage memory. (In particular, we wouldn't want to prevent the use of memory caches in the containers, which could prevent repeated allocate/deallocate cycles in some uses.)

Note that making it work only matters to generic packages; we don't have a mechanism to change the default pool for normal packages.

We didn't do this as it's unclear that the benefits outweight the costs (in particular, in implementation constraints). The rule would eliminate one tool (custom pools) from the implementer's performance enhancement toolbox.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0184**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.11.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0184 Pool_of_Subpool returns null when called too early

Working Reference Number AI12-0145-1

Question

What does Pool_of_Subpool return if Set_Pool_of_Subpool has never been called for the specific subpool handle? (Null.) 13.11.4(20/3) only says this about Pool_of_Subpool:

An access to the pool that a subpool belongs to can be obtained by calling Pool_of_Subpool with the subpool handle.

Summary of Response

Pool_of_Subpool returns null if called before Set_Pool_of_Subpool is called on the subpool handle.

Corrigendum Wording

Replace 13.11.4(20):

> Each subpool *belongs* to a single storage pool (which will always be a pool that supports subpools). An access to the pool that a subpool belongs to can be obtained by calling Pool_of_Subpool with the subpool handle. Set_Pool_of_Subpool causes the subpool of the subpool handle to belong to the given pool; this is intended to be called from subpool constructors like Create_Subpool. Set_Pool_of_Subpool propagates Program_Error if the subpool already belongs to a pool.

by:

> Each subpool *belongs* to a single storage pool (which will always be a pool that supports subpools). An access to the pool that a subpool belongs to can be obtained by calling Pool_of_Subpool with the subpool handle. Set_Pool_of_Subpool causes the subpool of the subpool handle to belong to the given pool; this is intended to be called from subpool constructors like Create_Subpool. Set_Pool_of_Subpool propagates Program_Error if the subpool already belongs to a pool. If Set_Pool_of_Subpool has not yet been called for a subpool, Pool_of_Subpool returns **null**.

Discussion

There are at least two plausible ways for such a call to happen:

(1) Create_Subpool (or other constructor) forgets to call Set_Pool_of_Subpool. (Equivalently, Pool_of_Subpool is called within a subpool constructor before the call to Set_Pool_of_Subpool).

(2) A pool creator visibly extends Root_Subpool, and then:

> My_Subpool_Object : My_Subpool_Type; My_Subpool : Subpool_Handler := My_Subpool_Object'access; ... Pool_of_Subpool(My_Subpool) ... -- ??
>
> Equivalently, the user extends Root_Subpool themselves.
>
> Pool creators aren't supposed to extend Root_Subpool visibly (they're supposed to do it in the package body) and users aren't supposed to extend these types at all, but we have no way to enforce intended use in this case.

It appears that the intent was for Pool_of_Subpool to return null in such a case. Two reasons why: (A) There is no "not null" on the return type, while we used that on almost every other access type in this package. Thus we expected "null" to be returned in some case, and this is the only case where that would have been possible. (B) Pool_of_Subpool is used in a Pre'Class in the package, and it seems unlikely that we'd want failure of that precondition to raise anything other than Assertion_Error.

In addition, the existing implementation returns null in this case. For all of these reasons, we add wording to say that null is returned if Set_Pool_of_Subpool has not yet been called.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0185**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.11.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0185 Bad subpool implementations

Working Reference Number AI12-0142-1

Question

We have 13.11(21) to say that compilers do not have to worry about malfunctioning pool implementations. However, this wording does not cover any memory allocated by Allocate_From_Subpool (as in that case, Allocate is never called). Should we have such wording? (Yes.)

Summary of Response

Add erroneous execution wording for subpools.

Corrigendum Wording

Insert after 13.11.4(31):

Unless overridden, Default_Subpool_for_Pool propagates Program_Error.

the new paragraph:

Erroneous Execution

If Allocate_From_Subpool does not meet one or more of the requirements on the Allocate procedure as given in the Erroneous Execution rules of 13.11, then the program execution is erroneous.

Discussion

13.11(21) is cleverly worded such that problems caused by the implementation of Deallocate or external action (as could happen if the pool type is not encapsulated) are blamed on the implementation of Allocate and thus also can cause erroneous execution. Specifically, such problems can cause the allocated memory to be used for some other purpose while the pool element still exists.

For allocation from a subpool, Allocate need never be called. If Allocate is never called, 13.11(21) does not apply. But of course Allocate_From_Subpool needs the same sort of requirements that Allocate does.

13.11.4(21/3) attempts to apply those requirements to Allocate_From_Subpool, but it fails to state the consequences. Thus we add a sentence under the proper heading to do that.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0186**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **13.11.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0186 Error in Storage Pool example

Working Reference Number AI12-0134-1

Question

The component Storage in 13.11.6(11/3) is declared thusly:

Storage : Storage_Array (0 .. Pool_Size-1);

This is illegal by 3.8(12/3), as the discriminant does not stand alone. Should this be fixed? (Yes.)

Summary of Response

The storage component in the example of 13.11.6(11/3) can't subtract 1.

Corrigendum Wording

Replace 13.11.6(11):

```
type Mark_Release_Pool_Type (Pool_Size : Storage_Count) is new
   Subpools.Root_Storage_Pool_With_Subpools with record
   Storage         : Storage_Array (0 .. Pool_Size-1);
   Next_Allocation : Storage_Count := 0;
   Markers         : Subpool_Array;
   Current_Pool    : Subpool_Indexes := 1;
end record;
```

by:

```
type Mark_Release_Pool_Type (Pool_Size : Storage_Count) is new
   Subpools.Root_Storage_Pool_With_Subpools with record
   Storage         : Storage_Array (0 .. Pool_Size);
   Next_Allocation : Storage_Count := 0;
   Markers         : Subpool_Array;
   Current_Pool    : Subpool_Indexes := 1;
end record;
```

Discussion

The subtraction is indeed illegal. We just removed it, as doing so only wastes a single storage unit (unlikely to be significant). Alternatively, we could have rewritten the pool to use a 1-based component. But that would probably introduce other bugs (particular in the alignment calculation, which was wrong in a previous version). Thus we didn't do that.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0187**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.13.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0187 Stream-oriented aspects

Working Reference Number AI12-0121-1

Question

13.13.2(38/3) says that the stream-oriented attributes can be specified with an attribute_definition_clause, and then it goes on to provide various legality rules for such specification. Nothing is mentioned about the rules for specification with an aspect_specification. Do the same rules apply? (Yes.)

Second, the rules prevent specifying a stream attribute for an interface as anything other than a null procedure. This applies to class-wide attributes as well as specific attributes. However, the reason for this restriction (inheritance issues) does not apply to overriding a class-wide stream attribute of an interface, since such attributes are never inherited. Should this be allowed? (Yes.)

Summary of Response

The stream-oriented attributes can be specified by an aspect specification, and the same legality rules apply. Class-wide attributes for interfaces can be specified as any appropriate nonabstract subprogram.

Corrigendum Wording

Replace 13.13.2(38):

> The stream-oriented attributes may be specified for any type via an attribute_definition_clause. The subprogram name given in such a clause shall statically denote a subprogram that is not an abstract subprogram. Furthermore, if a stream-oriented attribute is specified for an interface type by an attribute_definition_clause, the subprogram name given in the clause shall statically denote a null procedure.

by:

> The stream-oriented attributes may be specified for any type via an attribute_definition_clause. Alternatively, each of the specific stream-oriented attributes may be specified using an aspect_specification on any type_declaration other than an incomplete_type_declaration, with the aspect name being the corresponding attribute name.
>
> The subprogram name given in such an attribute_definition_clause or aspect_specification shall statically denote a subprogram that is not an abstract subprogram. Furthermore, if a specific stream-oriented attribute is specified for an interface type, the subprogram name given in the attribute_definition_clause or aspect_specification shall statically denote a null procedure.

Discussion

AI12-0106-1 addresses the problem of specifying class-wide stream attributes with aspect specifications, so we do not mention specifying those as aspects in this wording change.

The reason that interfaces only allow null procedures for (specific) stream attributes is explained in AARM 13.13.2(38.b/2). In particular, stream attributes of interfaces do not participate in extensions, unless the interface is the parent type (as opposed to a progenitor type). If the interfaces could have significant attributes, then the order of declarations could be significant (which violates an important design principle of interfaces).

However, class-wide stream attributes are never used in extensions, or inherited in any other way. So it is OK to replace them; we don't need to make such overrides illegal.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0188**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **13.13.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0188 Formal derived types and stream attribute availability

Working Reference Number AI12-0030-1

Question

Consider this example:

```
procedure Foo
   (S : not null access Ada.Streams.Root_Stream_Type'Class) is
    type Lim is limited null record;

    type Dlim is new Lim; -- does not inherit Lim'Read

    procedure Read_Lim
      (Stream : not null access Ada.Streams.Root_Stream_Type'Class;
       Item   : out Lim) is null;

    for Lim'Read use Read_Lim;
    Lim_Var : Lim;
    Dlim_Var : Dlim;

    generic
       type Flim is new Lim;
    procedure G;
    procedure G is
       Flim_Var : Flim;
    begin
       Flim'Read (S, Flim_Var); -- Legal? (Yes.)
    end G;
    procedure I is new G (Dlim);
begin
    Lim'Read (S, Lim_Var);
    Dlim'Read (S, Dlim_Var); -- Illegal.
    I; -- Legal, but raises Program_Error when called.

end;
```

Is the use of Flim'Read legal? (Yes.) If it is legal, are the dynamic semantics well defined? (Yes; when the procedure I invokes Dlim'Read, Program_Error is raised).

Assuming that stream attribute availability is a "characteristic", it seems that 12.5.1(20/3) applies and the use within the generic body is legal. This means we need to define the meaning of the corresponding construct in the instance.

The "even if it is never declared" wording in 12.5.1(21/3) handles a similar case, the case where a generic formal type promises a primitive operation that the corresponding actual type lacks. Do we need to add some analogous wording to handle this case? (No, but wording is added to ensure that the dynamic behavior of streaming attributes is defined in all cases including, for example, a task type with no user-defined streaming operations).

Summary of Response

For an untagged formal derived type, stream attributes (at runtime, in an instance) are those of the actual type. Those of the ancestor type do not reemerge.

Corrigendum Wording

Replace 13.13.2(49):

> An attribute_reference for one of the stream-oriented attributes is illegal unless the attribute is available at the place of the attribute_reference. Furthermore, an attribute_reference for *T*'Input is illegal if *T* is an abstract type.

by:

An attribute_reference for one of the stream-oriented attributes is illegal unless the attribute is available at the place of the attribute_reference. Furthermore, an attribute_reference for *T*'Input is illegal if *T* is an abstract type. In addition to the places where Legality Rules normally apply (see 12.3), these rules also apply in the private part of an instance of a generic unit.

Unless inherited from a parent type, if any, for an untagged type having a task, protected, or explicitly limited record part, the default implementation of each of the Read, Write, Input, and Output attributes raises Program_Error and performs no other action.

Discussion

The answer to the legality question is clear: characteristics of all sorts come from the ancestor type, not the actual type, so of course the attribute is legal and it will call the attribute of the ancestor type.

However, the wording of the Standard is definitely not clear on this point. "Availability of stream attributes" is not a characteristic of a type (at least it is not listed in 3.4, which is the definition of a "characteristic"). So the existing rules don't cover this case.

We could try to define "availability" as a "characteristic" (by adding a new bullet at 3.4(15), but that most likely would lead to conflicts as 7.3.1 defines how characteristics are inherited for private types, while 13.13.2 does so for "availability". These are similar, but probably not precisely the same, and we'd have to reword 13.13.2 in order to avoid confusion.

Alternatively, we could just throw some text and add a new rule after 12.5.1(20/3) specifically to say that "availability of stream attributes" is the same as that of the ancestor type, and the stream attribute called is that of the ancestor type. But that just adds a lot of text to say something obvious.

It's rare to see a derived untagged formal type, as such types can only match derived untagged types -- which is very limiting, especially given that untagged derivations themselves aren't that common (it's usually better to declare a new type). As such, it's not critical to have the Standard wording perfect in this area. So we simply add a To Be Honest note so there is no doubt of the intent.

However, that leaves the dynamic semantics. It's clear from 13.1(11/3) that the stream attributes (which are operational aspects) used in the instance are those of the actual type. But in this case, the actual type has no stream attributes defined. Thus we have to define what the attributes do in this case.

We've chosen to have them raise Program_Error. This means that the example given in the question compiles successfully and raises Program_Error at runtime. Raising Program_Error is fine - we just want the construct to be well defined and reasonably easy to implement. For an implemention which macro-expands instantiations, the Program_Error case is identifiable at compile-time.

The "In addition" boilerplate isn't really necessary, but consider modifying the example so that the generic is a generic package, not a generic procedure, and the problematic call to Flim'Read is replaced with a call to Flim'Input in the private part of the generic. Without the boilerplate (and with the rest of the change) this raises Program_Error. With the boilerplate, this gets caught at compile time, which is preferable.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0189**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **13.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0189 Expression functions that are completions in package specifications

Working Reference Number AI12-0103-1

Question

Consider the following:

```
package P is
   type Priv is private;

   Empty_Priv : constant Priv;

   function Is_Empty (A : Priv) return Boolean; -- (1)

private

   type Priv is record
      Len : Natural;
      Data: Natural;
   end record;

   -- Completions still need checking (call case):
   function Is_Empty (A : Priv) return Boolean is
      (A = Empty_Priv or else A.Len = 0);     -- (2)
      -- Not used in this package, so not frozen until the end of the package.

   Empty_Priv : constant Priv := (Len => 0, Data => 0); -- (3) Freezes Priv.
```

end P;

Is_Empty (2) is an expression function representing a completion, but 13.14(3/3) only applies in declarative_parts, while this is a package specification. So the expression function Is_Empty does not cause freezing at (2).

The completion of Empty_Priv at (3) is OK, as there is no requirement that Is_Empty uses only complete types.

Erasing the declaration of Is_Empty at (1) has no effect.

This seems to be what we want. If 13.14(3/3) did apply to package specifications, then the completion (2) would be illegal (as it is using an incomplete deferred constant), but it could be made legal by erasing the declaration at (1).

So, which freezing rules apply to expression_functions used as completions? (See below.)

Summary of Response

Expression functions that are a completion freeze their expression but do not freeze anything else (unlike regular bodies). Other expression functions don't freeze anything at all.

Similarly, null procedures never cause freezing, even if they are completions.

Corrigendum Wording

Replace 13.14(3):

> The end of a declarative_part, protected_body, or a declaration of a library package or generic library package, causes *freezing* of each entity and profile declared within it, except for incomplete types. A noninstance body other than a renames-as-body causes freezing of each entity and profile declared before it within the same declarative_part that is not an incomplete type; it only causes freezing of an incomplete type if the body is within the immediate scope of the incomplete type.

by:

The end of a declarative_part, protected_body, or a declaration of a library package or generic library package, causes *freezing* of each entity and profile declared within it, except for incomplete types. A proper_body, body_stub, or entry_body causes freezing of each entity and profile declared before it within the same declarative_part that is not an incomplete type; it only causes freezing of an incomplete type if the body is within the immediate scope of the incomplete type.

Insert after 13.14(5):

- The occurrence of a generic_instantiation causes freezing, except that a name which is a generic actual parameter whose corresponding generic formal parameter is a formal incomplete type (see 12.5.1) does not cause freezing. In addition, if a parameter of the instantiation is defaulted, the default_expression or default_name for that parameter causes freezing.

the new paragraph:

- At the occurrence of an expression_function_declaration that is a completion, the expression of the expression function causes freezing.

Discussion

The general principle is that things that might occur in a package specification freeze as little as is required to prevent semantic problems. In particular, renames-as-bodies are excluded from the "freeze everything" rules (despite acting as a body).

This is necessary so that the occurrence of one of these implicit bodies doesn't freeze unrelated entities from outer scopes.

Moreover, it's clear that 13.14(3/3) does not apply to package specifications. The reason for the rule, as explained by the following AARM notes, is that we want interchangeability of bodies (of all kinds) with body stubs (which have to be freezing, because we can't know their contents). However, body stubs are not allowed in package specifications. So there is no need to have rules that emulate having those.

Additionally, there is a usability issue. Freezing is mysterious to users (one user said they believed that it was invented to cover up compiler bugs!) As such, we should require as little freezing as possible to avoid semantic problems.

The primary argument for expression functions as completions causing freezing is for consistency with other bodies. But that is misleading, as renames-as-bodies are not freezing. So changing from a renames-as-body to an expression function to an regular body has to change freezing somewhere. Specifically, all of the following have essentially the same effect (assume Bar is a function, and these Foos are all completions):

```
function Foo return Integer renames Bar; -- Never freezing

function Foo return Integer is (Bar); -- Freezes only Bar (see below)

function Foo return Integer is        -- Always freezes
begin
    return Bar;
end Foo;
```

Clearly, there is going to be a wart of some sort in any case. Looking at all kinds of bodies (semantic font), we have a continuum of kinds of bodies: null procedures as completions; renames-as-body; expression functions as completions; generic instantiations; normal bodies ("proper_body" and "entry_body"); body stubs.

It's clearly never going to be the case that all of these have the same freezing rules. Thus we simply have the existing rule (13.14(3/3) only apply to the last two, and change the AARM notes accordingly.

Unfortunately, this is not quite the end of the story. Steve Baird pointed out that a modified version of the example in AARM 13.14(10.i-q/3) causes trouble:

```
package Pack is
   type Flub is range 0 .. 100;
```

```
      function Foo (A : in Natural) return Natural;
      type Bar is access function Foo (A : in Natural) return Natural;
      P : Bar := Foo'access; -- (A)
      function Foo (A : in Natural) return Natural is
         (A + Flub'Size); -- (B)
      Val : Natural := P.all(5); -- (C)
   end Pack;
```

13.14(10.3/3) does not apply to (A) (since the declaration of Foo is not an expression function). There is no rule (intentionally) to freeze the expressions of an expression function, so (B) does not freeze anything. However, then, the call at (C) is using a property of an unfrozen type.

Moreover, note that the call is after the body of Foo, so it passes elaboration checks.

We briefly considered changing elaboration checks to prevent this problem, but it would change something simple (elaboration checks) to fix something that is already complex (freezing).

Thus, we have adopted a rule that an expression function as completion freezes its expression (so it can be immediately called, as it is elaborated).

We could have complicated the rule so that it only applies when 'Access is taken of the subprogram within the immediate scope, but that would be an unusual context dependency for a freezing rule.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0190**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0190 Freezing of renames-as-body

Working Reference Number AI12-0132-1

Question

AI12-0103-1 appeals to an analogy with renames-as-body in order to determine how freezing for expression functions should work.

Unfortunately, an example using renames-as-body has a similar problem as the one described in AI12-0103-1 for expression functions:

```
package Pack is
   type Flub is range 0 .. 100;
   function Foo (A : in Natural) return Natural;
   function Bar (A : in Natural) return Natural is
      (A + Flub'Size); -- (A)
   type Fum is access function Foo (A : in Natural) return Natural;
   P : Fum := Foo'access; -- (B)
   function Foo (A : in Natural) return Natural renames Bar; -- (C)
   Val : Natural := P.all(5); -- (D)
end Pack;
```

(A) doesn't freeze anything. (B) does not trigger 13.14(10.1/3), since Foo is not an expression function. (C) doesn't appear to freeze anything. Finally, the call at (D) is legal and passes elaboration checks, but the value of the result depends on a representation aspect of the unfrozen type Flub.

This hole needs to be plugged somehow, right? (Yes.)

Summary of Response

A renames-as-body freezes the expression of any expression function that it renames.

Corrigendum Wording

Insert after 13.14(5):

- The occurrence of a generic_instantiation causes freezing, except that a name which is a generic actual parameter whose corresponding generic formal parameter is a formal incomplete type (see 12.5.1) does not cause freezing. In addition, if a parameter of the instantiation is defaulted, the default_expression or default_name for that parameter causes freezing.

the new paragraph:

- At the occurrence of a renames-as-body whose *callable_entity*_name denotes an expression function, the expression of the expression function causes freezing.

Discussion

There isn't any rule in Ada 2012 that causes the freezing of the name of a renamed entity.

We didn't worry about freezing these in Ada 95 and Ada 2005 because any problems would be detected by a failed elaboration check. For instance, an Ada 95 example like the above could have been:

```
package Pack2 is
   type Flub is range 0 .. 100;
   function Foo (A : in Natural) return Natural;
   function Bar (A : in Natural) return Natural; -- (A)
   type Fum is access function Foo (A : in Natural) return Natural;
   P : Fum := Foo'access; -- (B)
   function Foo (A : in Natural) return Natural renames Bar; -- (C)
   Val : Natural := P.all(5); -- (D)
end Pack2;
```

In this case, nothing freezes Bar, but the call at (D) will raise Program_Error, so there will not be any possibility of evaluating something unfrozen (like Flub). If Bar had been an instance, it would have frozen everything before it.

Thus, it appears that this bug is also related to the introduction of expression functions (for which neither the elaboration check nor freezing work).

The proposed solution applies only to expression functions, as any other solution would potentially be incompatible.

In the example in the question, the proposed rule means that (C) causes the expression of (A) to freeze. Then the call at (D) is safe because Flub has been frozen.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0191**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **A**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0191 Implicit objects are considered overlapping

Working Reference Number AI12-0052-1

Question

We all know that calling Put on the same file from multiple tasks is not expected to work. The justification is that A(3) says that language-defined operations have to work when "nonoverlapping" objects are involved. Clearly a file "overlaps" with itself.

However, the situation is murkier when one of the default files is involved. The default files are not parameters to the operations in question, so the A(3) rules don't (obviously) apply to them.

While it might be possible to tease out that these are the same objects, it would better to mention this explicitly, right? (Yes.)

Summary of Response

The default input and output files of Text_IO are considered implicit parameters to the associated routines for the purposes of determining whether A(3) applies.

A(3) applies to any pair of concurrent calls to language-defined subprograms, not just to calls to the same subprogram.

Corrigendum Wording

Replace A(3):

> The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on the same subprogram perform as specified, so long as all parameters that could be passed by reference denote nonoverlapping objects.

by:

> The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on any language-defined subprogram perform as specified, so long as all parameters that could be passed by reference denote nonoverlapping objects.
>
> For the purpose of determining whether concurrent calls on text input-output subprograms are required to perform as specified above, when calling a subprogram within Text_IO or its children that implicitly operates on one of the default input/output files, the subprogram is considered to have a parameter of Current_Input or Current_Output (as appropriate).

Discussion

We make a wording change to A(3) to make it crystal clear that it applies to any pair of calls to language-defined subprograms from different tasks. In particular, we want the wording to ensure that:

```
task T1 is
begin
    Put (A_File, "Text");
end T1;

task T2 is
begin
    Put_Line (A_File, "More Text");
end T2;
```

is not required to work (being unsafe use of shared variables), while

```
task T1 is
begin
    Put (A_File, "Text");
end T1;
```

```
task T2 is
begin
    Put_Line (B_File, "More Text");
end T2;
```

is always required to work (assuming A_File and B_File are different file objects, not renamings or formal parameters mapped to the same actual).

There was concern that wording is also needed for "Current_Error", but as it is never a default file for any operation in Ada.Text_IO, the wording given for the insertion after A.10.3(21) does not need to mention it.

We mention "or its children" in the Text_IO wording so that calling a routine defined in a package like Text_IO.Bounded_IO is covered appropriately.

We considered applying similar rules for the cases of the current default directory and environment variables, where the "state" would considered an implicit parameter. However, this would make these packages harder to use in a multitasking system. Moreover, the calls are likely to be expensive anyway (if this state is managed by the underlying target OS), and it is quite likely that the target OS already provides some guarantees for concurrent access.

Therefore, we require these packages to "work" when called concurrently. Extra locking might be necessary (especially when the packages are implemented without using an underlying OS).

We don't believe that there are any other "global" data structures used by the definition of language-defined packages that could require locking. Clocks and locales only can be read (if setting is provided, it's not via a language-defined subprogram, and thus isn't covered by A(3)). Of course, implementations can do what they want with implementation-defined routines.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0192**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0192 Overlapping objects designated by access parameters are not thread-safe

Working Reference Number AI12-0114-1

Question

A(3) mandates that "The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on {any language-defined}[the same] subprogram perform as specified, so long as all parameters that could be passed by reference denote nonoverlapping objects."

The rule does not apply when overlapping objects designated by access parameters overlap. Hence such calls would need to be made reentrant, i.e. thread-safe, in all cases. Is this intended? [No.]

Summary of Response

A(3) not only applies to nonoverlapping objects denoted by parameters passed by reference but also to nonoverlapping objects designated by parameters of access type.

Corrigendum Wording

Replace A(3):

> The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on the same subprogram perform as specified, so long as all parameters that could be passed by reference denote nonoverlapping objects.

by:

> The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on any language-defined subprogram perform as specified, so long as all objects that are denoted by parameters that could be passed by reference or designated by parameters of an access type are nonoverlapping.

Response

A(3) need to be extended so that overlapping objects are excluded, whether they're passed by reference or designated by an access value.

This solution is preferable to special-case rules that deal with each problematic interface individually. Luckily the rule is written such that in practice it will not impact any subprograms in which access parameters can access nonoverlapping objects safely.

Discussion

Unfortunately, the issues that surround A(3), e.g., the alteration of file handles as part of IO operations, not only apply to parameters passed by reference, but also to parameters of access types passed by copy or copy-in/copy-out.

Most notably, 'Read, 'Write, 'Input and 'Output operate on access Ada.Streams.Root_Stream_Type'Class, causing the very same issues that accompany file handles in IO operations.

Other examples in predefined packages include: The types System.Storage_Pools.Subpools.Subpool_Handle, Interfaces.C.Strings.char_array_access and Interfaces.C.Pointers.Pointer used for parameters of subprograms, or the declaration of the parameter Interfaces.C.Pointers.Do_RPC.Result as an IN parameter.

The likelihood of the user encountering the problem in user code is demonstrated in the RM by the procedure Tapes.Copy using access parameters of mode IN with the clear intent to alter the object designed by one of the parameters.

While potential data races in user-written code are the responsibility of the user, the above examples show that the predefined packages also include interfaces that would be affected by data races when the same object is designated by access parameters of concurrent calls.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0193**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.4.11**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0193 UTF_Encoding.Conversions and overlong characters on input

Working Reference Number AI12-0088-1

Question

A.4.11(50/3) says that "For UTF-8, no overlong encoding is returned". It does not say what happens if there is a character with overlong encoding on input.

Overlong encodings used to be tolerated by 10646, but are now considered invalid. Therefore, an overlong input should raise Encoding_Error per A.4.11(54/3). Should this be enforced? (No.)

Summary of Response

An overlong encoding in the input to Encode and Convert should not raise Encoding_Error.

Corrigendum Wording

Replace A.4.11(54):

- By a Decode function when a UTF encoded string contains an invalid encoding sequence.

by:

- By a Convert or Decode function when a UTF encoded string contains an invalid encoding sequence.

Replace A.4.11(55):

- By a Decode function when the expected encoding is UTF-16BE or UTF-16LE and the input string has an odd length.

by:

- By a Convert or Decode function when the expected encoding is UTF-16BE or UTF-16LE and the input string has an odd length.

Discussion

You cannot optimize away a conversion from UTF-8 to UTF-8 because there must be a check for an overlong encoding, in order to remove those encodings from the result. The only known Ada 2012 implementation at this writing fails to do this conversion, so the output violates A.4.11(50/3).

We considered declaring that overlong encodings are invalid (so far as this package is concerned), but that's unfriendly to users. If an overlong encoding appears in some text that they are processing, an Encoding_Error exception would prevent getting any results, even though there is a well-defined meaning to the text. It's better to accept the overlong encoding and convert it to the standard form.

One can detect overlong encodings by doing a Convert from UTF-8 to UTF-8; if the length changes, the input contains an overlong encoding.

Note that A.4.11(54/3) and A.4.11(55/3) are written to apply only to Decode functions, but should also apply to Convert functions (both take encoded input).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0194**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **A.8.1; A.8.2; A.8.4; A.10.3; A.12.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0194 All I/O packages should have Flush

Working Reference Number AI12-0130-1

Question

Text_IO and Stream_IO have procedure Flush to ensure that internal buffers are "flushed". However, Sequential_IO and Direct_IO do not. This makes it impossible to ensure that a write using one of these packages has completed. Should these routines be added? (Yes.)

Summary of Response

Flush is added to Sequential_IO and Direct_IO.

Corrigendum Wording

Insert after A.8.1(10):

```
function Is_Open(File : in File_Type) return Boolean;
```

the new paragraph:

```
procedure Flush (File : in File_Type);
```

Insert after A.8.2(28):

Returns True if the file is open (that is, if it is associated with an external file); otherwise, returns False.

the new paragraphs:

```
procedure Flush(File : in File_Type);
```

The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file. For a direct file, the current index is unchanged; for a stream file (see A.12.1), the current position is unchanged.

The exception Status_Error is propagated if the file is not open. The exception Mode_Error is propagated if the mode of the file is In_File.

Insert after A.8.4(10):

```
function Is_Open(File : in File_Type) return Boolean;
```

the new paragraph:

```
procedure Flush (File : in File_Type);
```

Replace A.10.3(21):

The effect of Flush is the same as the corresponding subprogram in {Sequential_IO (see A.8.2)}[Streams.Stream_IO (see A.12.1)]. If File is not explicitly specified, Current_Output is used.

by:

The effect of Flush is the same as the corresponding subprogram in Sequential_IO (see A.8.2). If File is not explicitly specified, Current_Output is used.

Replace A.12.1(28):

The subprograms given in subclause A.8.2 for the control of external files (Create, Open, Close, Delete, Reset, Mode, Name, Form, and Is_Open) are available for stream files.

by:

The subprograms given in subclause A.8.2 for the control of external files (Create, Open, Close, Delete, Reset, Mode, Name, Form, Is_Open, and Flush) are available for stream files.

Delete A.12.1(28.6):

> The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file or changing the position. Mode_Error is propagated if the mode of the file is In_File.

Discussion

Consistency suggests that Flush should be in all of the I/O packages, not just two.

It's thought that Flush wasn't included in Sequential_IO and Direct_IO because these packages don't use any buffering. But the definition of those packages only specify the behavior relative to the internal file; what the target system does with the external file isn't specified. In particular, it could use buffering whether or not it makes sense to the Ada program. Most implementations of Flush also call a target OS function to ensure that any pending writes are completed. It's odd that this functionality isn't available in Sequential_IO and Direct_IO; this lack makes it harder to write cooperating programs using these I/O packages.

It's possible to use Stream_IO (which has Flush) to emulate Sequential_IO and Direct_IO. But such code is necessarily more complicated and error-prone than using the original packages (especially for Direct_IO, which requires calculating a positioning location, an operation that is easy to get wrong).

Since this is inconsistent for no obvious reason, and the routine has value for all of the I/O packages, we add the missing procedures. We moved the definition of Flush to the Sequential_IO subclause, as that is where all of the existing file management subprograms are defined, and it would be strange to have all but one defined there. (Flush was defined in Stream_IO in Ada 95-Ada 2012.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0195**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.12.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0195 Stream_IO.File_Type has Preelaborable_Initialization

Working Reference Number AI12-0102-1

Question

As a last minute change to Ada 2012, we made Stream_IO preelaborable so that it can be used for logging and persistence in preelaborable packages (especially useful in distributed systems).

There was an ACATS test submitted for this, which starts:

```
with Ada.Streams.Stream_IO;
package CXAC007_0 with Preelaborate is

   File : Ada.Streams.Stream_IO.File_Type;

   function Is_Open return Boolean is (Ada.Streams.Stream_IO.Is_Open (File));
end CXAC007_0;
```

GNAT compiles this without complaint.

Unfortunately, I believe this should be illegal. The declaration of File clearly is covered by 10.2.1(9/3): "The creation of an object that is initialized by default, if its type does not have preelaborable initialization."

Stream_IO defines File_Type (A.12.1(5)) as:

```
type File_Type is limited private;
```

10.2.1(11.2/2) says that "A partial view of a private type ... has preelaborable initialization if and only if the pragma Preelaborable_Initialization has been applied to them."

No matter how hard I try, I can't see "pragma Preelaborable_Initialization(File_Type);" in A.12.1(5) or anywhere else in Stream_IO for that matter. We clearly did not consider this when we discussed AI05-0283-1.

The majority of other language-defined private types in Pure and Preelaborated packages have Preelaborable_Initialization (P_I). Types like Controlled, Root_Stream_Type, and Address all have P_I.

Should this type have P_I, making this example legal? (Yes.)

Summary of Response

Ada.Stream_IO.File_Type has Preelaborable_Initialization.

Corrigendum Wording

Replace A.12.1(5):

```
type File_Type is limited private;
```

by:

```
type File_Type is limited private;
pragma Preelaborable_Initialization(File_Type);
```

Discussion

It would be unusual to be unable to declare a library-level Ada.Stream_IO.File_Type object in a preelaborated package. One could still use the package (File_Types would have to be declared within subprograms), so this isn't an open-and-shut case for a mistake. But it seems more likely to be a mistake, especially as types like Controlled have Preelaborable_Initialization (P_I). Indeed, nearly all private types in language-defined Pure and Preelaborated packages have P_I.

So we declare Ada.Stream_IO.File_Type to have P_I.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0196**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **A.18; A.18.11; A.18.12; A.18.13; A.18.14; A.18.15; A.18.16; A.18.17; A.18.18**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0196 Accessibility checks for indefinite elements of containers

Working Reference Number AI12-0035-1

Question

There appears to be a hole in the indefinite container semantics. Consider the following case: The Indefinite_Holders container is instantiated with a class-wide type, and a value of an object of a more deeply nested tagged type is saved in a container of the outer-level instance via a call to Replace_Element. If this procedure is implemented in Ada, using an allocator, the allocator would fail a run-time accessibility check, thus raising Program_Error. This appears to be the behavior we want, as otherwise it would be possible to store a value of a type whose scope has exited within a longer-lived container, and that surely shouldn't be permitted. (Such an object within a container could be referred to as "dangling", in that its tag refers to a type that no longer exists.)

Should this be fixed? (Yes.)

Summary of Response

Certain operations of instances of the indefinite container packages require accessibility checks to prevent dangling references. This is specified in terms of the checks that would be required for executing an initialized allocator for a notional access type with the designated type of the element type of the instance.

Corrigendum Wording

Insert after A.18(5):

When a formal function is used to provide an ordering for a container, it is generally required to define a strict weak ordering. A function "<" defines a *strict weak ordering* if it is irreflexive, asymmetric, transitive, and in addition, if $x < y$ for any values x and y, then for all other values z, ($x < z$) or ($z < y$).

the new paragraphs:

Static Semantics

Certain subprograms declared within instances of some of the generic packages presented in this clause are said to *perform indefinite insertion*. These subprograms are those corresponding (in the sense of the copying described in subclause 12.3) to subprograms that have formal parameters of a generic formal indefinite type and that are identified as performing indefinite insertion in the subclause defining the generic package.

If a subprogram performs indefinite insertion, then certain run-time checks are performed as part of a call to the subprogram; if any of these checks fail, then the resulting exception is propagated to the caller and the container is not modified by the call. These checks are performed for each parameter corresponding (in the sense of the copying described in 12.3) to a parameter in the corresponding generic whose type is a generic formal indefinite type. The checks performed for a given parameter are those checks explicitly specified in subclause 4.8 that would be performed as part of the evaluation of an initialized allocator whose access type is declared immediately within the instance, where:

- the value of the qualified_expression is that of the parameter; and
- the designated subtype of the access type is the subtype of the parameter; and
- finalization of the collection of the access type has started if and only if the finalization of the instance has started.

Insert after A.18.11(8):

- The actual Element parameter of access subprogram Process of Update_Element may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations "&", Append, Insert, Prepend, Replace_Element, and To_Vector that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Insert after A.18.12(7):

- The actual Element parameter of access subprogram Process of Update_Element may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations Append, Insert, Prepend, and Replace_Element that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Insert after A.18.13(8):

- The actual Element parameter of access subprogram Process of Update_Element may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations Include, Insert, Replace, and Replace_Element that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Insert after A.18.14(8):

- The actual Element parameter of access subprogram Process of Update_Element may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations Include, Insert, Replace, and Replace_Element that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Insert after A.18.15(4):

- The actual Element parameter of access subprogram Process of Update_Element_Preserving_Key may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations Include, Insert, Replace, Replace_Element, and To_Set that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Insert after A.18.16(4):

- The actual Element parameter of access subprogram Process of Update_Element_Preserving_Key may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations Include, Insert, Replace, Replace_Element, and To_Set that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Insert after A.18.17(7):

- The actual Element parameter of access subprogram Process of Update_Element may be constrained even if Element_Type is unconstrained.

the new paragraph:

- The operations Append_Child, Insert_Child, Prepend_Child, and Replace_Element that have a formal parameter of type Element_Type perform indefinite insertion (see A.18).

Replace A.18.18(39):

Returns a nonempty holder containing an element initialized to New_Item.

by:

Returns a nonempty holder containing an element initialized to New_Item. To_Holder performs indefinite insertion (see A.18).

Replace A.18.18(47):

> Replace_Element assigns the value New_Item into Container, replacing any preexisting content of Container. Container is not empty after a successful call to Replace_Element.

by:

> Replace_Element assigns the value New_Item into Container, replacing any preexisting content of Container; Replace_Element performs indefinite insertion (see A.18). Container is not empty after a successful call to Replace_Element.

Response

See summary.

Discussion

Randy wrote the following as background for this AI:

We want to be able to write the implementation of containers packages in Ada, and we expect the implementation to use allocators. As such, it would be odd if some of the reasons that those allocators could fail were not reflected in the specification.

A grocery list of considerations for this solution:

(1) The problem occurs for any indefinite container, and occurs anytime an element is created in a container. That is not just Replace_Element but also Insert_Element and Append_Element (and possibly others, I didn't check carefully). That means that there are a number of places that we need new rules.

(2) Indefinite containers are just a "simple" modification of the definite ones. Not sure if we have to modify the definite containers to deal with this or not (most of the complex rules are designed to apply to all of the container forms). In particular, I cannot be sure that it is impossible to cause the access discriminant check to fail for a definite subtype, as it applies to parts and constrained subtypes. Therefore, it would be safer to cover all containers, even though the check is unlikely to fail in a definite container.

(3) The idea of repeating the rules for accessibility for an allocator gives me uncontrollable shakes. We'd never get the right; they'd always be a version behind the allocator rules. We have to find some way to explain this in terms of a hypothetical allocator.

> The best I can do would be something like: "Program_Error is propagated by Replace_Element if an initialized allocator for the element value where the access type designates the element type and is declared at the point of the instantiation of the container package would fail an accessibility check." This is too hard to parse, I fear.

(4) We probably don't need any static rules for this problem. For an allocator of a generic formal type used in a generic body, these rules are checked only dynamically. The implementation of the container can (and ought to) avoid putting any allocators into the specification.

(5) This check might be very expensive to implement. But the implementability should be no easier or harder than the checks for any allocator given in a generic body. Whether *any* of these checks are really implementable is the subject of AI12-0016-1, and if we make changes based on that AI, they should apply here, too.

(6) It would be nice to have some sort of blanket rule that covers all container types and operations, but we rarely do that for the containers and it is not at all clear how it could be written in order to ensure that the right routines are covered and not any others.

The checks are only required on certain operations declared within certain instances of the indefinite containers. Specifically, they're needed when the actual types for the indefinite formal types (the Element_Type and Key_Type types) are class-wide types or indefinite subtypes with access discriminants, and for the operations that have parameters of those actual types that create and initialize container elements using

those parameters. Those are the cases where dangling references could potentially be created, and are precisely the cases where an implementation that uses access types and allocators would naturally apply the accessibility checks required for allocators in subclause 4.8.

Unfortunately there doesn't seem to be a simple way to characterize the set of operations (for example, it's a different set in each indefinite container), which is why the subclause that defines each of the indefinite containers gives a list of the operations that perform indefinite allocation.

Of course, the downside of specifically identifying the operations is that when any new such operations are added, the list will need to be updated to include them.

One might wonder whether it's possible to stream in something whose accessibility is too shallow for the container. The check is against the element type of the container instance, and the contents have to live at least as long as that element type. Thus, anything we can stream out has already been checked. And thus anything that we can stream in is already OK. (Recall that we only promise to be able to stream in stuff that we stream out, not anything in general.) And we don't promise any extra interoperability for streaming of indefinite containers. (We do for the bounded and unbounded forms, but that doesn't apply to the indefinite forms.) If there were an interoperability promise, a less nested instance could cause trouble.

Example

Here's an example of how such violations can occur for instantiations of indefinite containers with class-wide element types:

```
with Ada.Containers.Indefinite_Holders;
with Ada.Text_IO;  use Ada.Text_IO;

procedure Dangling_Container is

   package Pkg is

      type Abst is abstract tagged null record;

      procedure Print_Obj (Obj : Abst) is abstract;

   end Pkg;

   use Pkg;

   package Holder is
     new Ada.Containers.Indefinite_Holders (Element_Type => Abst'Class);

   Factory : Holder.Holder := Holder.Empty_Holder;

   procedure Set_Factory is

      type Deeper_Type is new Pkg.Abst with record
         C : Integer;
      end record;

      procedure Print_Obj (Obj : Deeper_Type);

      A : Deeper_Type := (C => 123);

      procedure Print_Obj (Obj : Deeper_Type) is
      begin
         Put_Line
           ("In Print_Obj for Deeper_Type, Obj.C =" & Integer'Image (Obj.C));

         A := Obj;  -- Assign to possibly nonexistent object; may damage stack
      end Print_Obj;

   begin
      Factory.Replace_Element (A);  -- Raises Program_Error because of new
rules
   end Set_Factory;

begin
```

```
      Set_Factory;

      Print_Obj (Factory.Element);  -- The program should never get this far!

      Put_Line ("Returned from call to out of scope primitive, ending program");
   end Dangling_Container;
```

and here's a program illustrating the access discriminant case:

```
   with Ada.Containers.Indefinite_Holders;
   with Ada.Text_IO;  use Ada.Text_IO;

   procedure Dangling_Acc_Discrim is

      type Acc_Discrim_Rec (D : access Integer) is record
         C : Integer;
      end record;

      package Holder is
        new Ada.Containers.Indefinite_Holders (Element_Type => Acc_Discrim_Rec);

      Factory : Holder.Holder := Holder.Empty_Holder;

      procedure Set_Factory is

         Aliased_Int : aliased Integer := 123;

         Nested_Obj : Acc_Discrim_Rec (D => Aliased_Int'access);

      begin
         Nested_Obj.C := 456;

         Factory.Replace_Element (Nested_Obj); -- Raises Program_Error because of
   new rules
      end Set_Factory;

      procedure Print_Obj (Obj : Acc_Discrim_Rec) is
      begin
         Put_Line ("In Print_Obj, Obj.D.all =" & Integer'Image (Obj.D.all)
                             & ", Obj.C =" & Integer'Image (Obj.C));

         Obj.D.all := Obj.D.all + Obj.C;
           -- Assignment to nonexistent object might damage stack
      end Print_Obj;

   begin
      Set_Factory;

      Print_Obj (Factory.Element);

      Put_Line ("After first call to Print_Obj");

      Print_Obj (Factory.Element);

      Put_Line ("After second call to Print_Obj");
```

end Dangling_Acc_Discrim;

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0197**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.18.2; A.18.3; A.18.4; A.18.7; A.18.10; A.18.18**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0197 Tampering checks are performed first

Working Reference Number AI12-0110-1

Question

What should happen when inserting into a full bounded container when tampering is prohibited? Should it cause Capacity_Error because the container is full, or Program_Error because it's tampering when tampering is prohibited? Or is this unspecified? (Program_Error.)

Summary of Response

Any needed tampering checks are performed before any other language-defined checks.

Corrigendum Wording

Replace A.18.2(97.1):

When tampering with cursors is *prohibited* for a particular vector object *V*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *V*, leaving *V* unmodified. Similarly, when tampering with elements is *prohibited* for a particular vector object *V*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *V* (or tamper with the cursors of *V*), leaving *V* unmodified.

by:

When tampering with cursors is *prohibited* for a particular vector object *V*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *V*, leaving *V* unmodified. Similarly, when tampering with elements is *prohibited* for a particular vector object *V*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *V* (or tamper with the cursors of *V*), leaving *V* unmodified. These checks are made before any other defined behavior of the body of the language-defined subprogram.

Replace A.18.3(69.1):

When tampering with cursors is *prohibited* for a particular list object *L*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *L*, leaving *L* unmodified. Similarly, when tampering with elements is *prohibited* for a particular list object *L*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *L* (or tamper with the cursors of *L*), leaving *L* unmodified.

by:

When tampering with cursors is *prohibited* for a particular list object *L*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *L*, leaving *L* unmodified. Similarly, when tampering with elements is *prohibited* for a particular list object *L*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *L* (or tamper with the cursors of *L*), leaving *L* unmodified. These checks are made before any other defined behavior of the body of the language-defined subprogram.

Replace A.18.4(15.1):

When tampering with cursors is *prohibited* for a particular map object *M*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *M*, leaving *M* unmodified. Similarly, when tampering with elements is *prohibited* for a particular map object *M*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *M* (or tamper with the cursors of *M*), leaving *M* unmodified.

by:

When tampering with cursors is *prohibited* for a particular map object *M*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *M*, leaving *M* unmodified. Similarly, when tampering with elements is *prohibited* for a particular map object *M*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with

the elements of *M* (or tamper with the cursors of *M*), leaving *M* unmodified. These checks are made before any other defined behavior of the body of the language-defined subprogram.

Replace A.18.7(14.1):

When tampering with cursors is *prohibited* for a particular set object *S*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *S*, leaving *S* unmodified. Similarly, when tampering with elements is *prohibited* for a particular set object *S*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *S* (or tamper with the cursors of *S*), leaving *S* unmodified.

by:

When tampering with cursors is *prohibited* for a particular set object *S*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *S*, leaving *S* unmodified. Similarly, when tampering with elements is *prohibited* for a particular set object *S*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *S* (or tamper with the cursors of *S*), leaving *S* unmodified. These checks are made before any other defined behavior of the body of the language-defined subprogram.

Replace A.18.10(90):

When tampering with cursors is *prohibited* for a particular tree object *T*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *T*, leaving *T* unmodified. Similarly, when tampering with elements is *prohibited* for a particular tree object *T*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *T* (or tamper with the cursors of *T*), leaving *T* unmodified.

by:

When tampering with cursors is *prohibited* for a particular tree object *T*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the cursors of *T*, leaving *T* unmodified. Similarly, when tampering with elements is *prohibited* for a particular tree object *T*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the elements of *T* (or tamper with the cursors of *T*), leaving *T* unmodified. These checks are made before any other defined behavior of the body of the language-defined subprogram.

Replace A.18.18(35):

When tampering with the element is *prohibited* for a particular holder object *H*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the element of *H*, leaving *H* unmodified.

by:

When tampering with the element is *prohibited* for a particular holder object *H*, Program_Error is propagated by a call of any language-defined subprogram that is defined to tamper with the element of *H*, leaving *H* unmodified. These checks are made before any other defined behavior of the body of the language-defined subprogram.

Discussion

Generally, we have been careful to define the order of the checks for the containers. In this case, there is no rule given.

Tampering occurs when one calls a subprogram that they're not allowed to call in the current context. As such this check should be early in the call, before more specific things happen.

For instance, consider inserting into a full bounded container when tampering with cursors is prohibited. In this case, the insertion is prohibited; the fact that the container is full is secondary. If the programmer increased the size of the container to get rid of the Capacity_Error, they shouldn't be greeted with a Program_Error from calling Insert in the wrong place.

An alternative to this wording would be to adopt the Pre- and Post-conditions proposed in AI12-0112-1. That seems like too much change for a corrigendum, but could be the long-term solution to this issue.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0198**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.18.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0198 Definition of node for tree container is confusing

Working Reference Number AI12-0078-1

Question

The introduction to the Tree container the definition of "node" is quite confusing, as it starts by saying that each node contains an element, but in fact the root node does not. Should this be clarified? (Yes.)

Summary of Response

The root node of a tree does not have an element.

Corrigendum Wording

Replace A.18.10(2):

> A multiway tree container object manages a tree of internal *nodes*, each of which contains an element and pointers to the parent, first child, last child, next (successor) sibling, and previous (predecessor) sibling internal nodes. A cursor designates a particular node within a tree (and by extension the element contained in that node, if any). A cursor keeps designating the same node (and element) as long as the node is part of the container, even if the node is moved within the container.

by:

> A multiway tree container object manages a tree of *nodes*, comprising of a *root node*, and a set of *internal nodes* each of which contains an element and pointers to the parent, first child, last child, next (successor) sibling, and previous (predecessor) sibling internal nodes. A cursor designates a particular node within a tree (and by extension the element contained in that node, if any). A cursor keeps designating the same node (and element) as long as the node is part of the container, even if the node is moved within the container.

Replace A.18.10(3):

> A *subtree* is a particular node (which *roots the subtree*) and all of its child nodes (including all of the children of the child nodes, recursively). There is a special node, the *root*, which is always present and has neither an associated element value nor any parent node. The root node provides a place to add nodes to an otherwise empty tree and represents the base of the tree.

by:

> A *subtree* is a particular node (which *roots the subtree*) and all of its child nodes (including all of the children of the child nodes, recursively). The root node is always present and has neither an associated element value nor any parent node; it has pointers to its first child and its last child, if any. The root node provides a place to add nodes to an otherwise empty tree and represents the base of the tree.

Response

(See !summary.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0199**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.18.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0199 Inconsistency in Tree container definition

Working Reference Number AI12-0069-1

Question

A.18.10(153/3) says:

Iterate calls Process.all with a cursor that designates each element in Container, starting with the root node and proceeding in a depth-first order.

whereas A.18.10(157/3) says:

Iterate returns an iterator object (see 5.5.1) that will generate a value for a loop parameter (see 5.5.2) designating each node in Container, starting with the root node and proceeding in a depth-first order.

The first does not include the root node in the iteration (it has no element), while the second does.

Should these work the same way? (Yes.)

Summary of Response

The root node is never visited by a container iterator.

Corrigendum Wording

Replace A.18.10(153):

> Iterate calls Process.**all** with a cursor that designates each element in Container, starting with the root node and proceeding in a depth-first order. Tampering with the cursors of Container is prohibited during the execution of a call on Process.**all**. Any exception raised by Process.**all** is propagated.

by:

> Iterate calls Process.**all** with a cursor that designates each element in Container, starting from the root node and proceeding in a depth-first order. Tampering with the cursors of Container is prohibited during the execution of a call on Process.**all**. Any exception raised by Process.**all** is propagated.

Replace A.18.10(155):

> If Position equals No_Element, then Constraint_Error is propagated. Otherwise, Iterate_Subtree calls Process.**all** with a cursor that designates each element in the subtree rooted by the node designated by Position, starting with the node designated by Position and proceeding in a depth-first order. Tampering with the cursors of the tree that contains the element designated by Position is prohibited during the execution of a call on Process.**all**. Any exception raised by Process.**all** is propagated.

by:

> If Position equals No_Element, then Constraint_Error is propagated. Otherwise, Iterate_Subtree calls Process.**all** with a cursor that designates each element in the subtree rooted by the node designated by Position, starting from the node designated by Position and proceeding in a depth-first order. Tampering with the cursors of the tree that contains the element designated by Position is prohibited during the execution of a call on Process.**all**. Any exception raised by Process.**all** is propagated.

Replace A.18.10(157):

> Iterate returns an iterator object (see 5.5.1) that will generate a value for a loop parameter (see 5.5.2) designating each node in Container, starting with the root node and proceeding in a depth-first order. Tampering with the cursors of Container is prohibited while the iterator object exists

(in particular, in the sequence_of_statements of the loop_statement whose iterator_specification denotes this object). The iterator object needs finalization.

by:

Iterate returns an iterator object (see 5.5.1) that will generate a value for a loop parameter (see 5.5.2) designating each element in Container, starting from the root node and proceeding in a depth-first order. Tampering with the cursors of Container is prohibited while the iterator object exists (in particular, in the sequence_of_statements of the loop_statement whose iterator_specification denotes this object). The iterator object needs finalization.

Replace A.18.10(159):

If Position equals No_Element, then Constraint_Error is propagated. Otherwise, Iterate_Subtree returns an iterator object (see 5.5.1) that will generate a value for a loop parameter (see 5.5.2) designating each element in the subtree rooted by the node designated by Position, starting with the node designated by Position and proceeding in a depth-first order. If Position equals No_Element, then Constraint_Error is propagated. Tampering with the cursors of the container that contains the node designated by Position is prohibited while the iterator object exists (in particular, in the sequence_of_statements of the loop_statement whose iterator_specification denotes this object). The iterator object needs finalization.

by:

If Position equals No_Element, then Constraint_Error is propagated. Otherwise, Iterate_Subtree returns an iterator object (see 5.5.1) that will generate a value for a loop parameter (see 5.5.2) designating each element in the subtree rooted by the node designated by Position, starting from the node designated by Position and proceeding in a depth-first order. If Position equals No_Element, then Constraint_Error is propagated. Tampering with the cursors of the container that contains the node designated by Position is prohibited while the iterator object exists (in particular, in the sequence_of_statements of the loop_statement whose iterator_specification denotes this object). The iterator object needs finalization.

Response

(See !summary.)

Discussion

AARM A.18.10(153.a/3) says that "Process is not called with the root node", so it's clear that not visiting the root node is intended for the procedure Iterate.

Additionally, both of the Iterate_Subtree forms say "each element", so the root node is not visited should it be passed to those routines.

It makes no sense for function Iterate to be different, especially as visiting the root node would require all loops to test for that before accessing the element (as the root node does not have an element and attempting to access the element raises an exception).

Thus we change the wording to say "each element" rather than "each node".

In looking at this, it was noted that the wording says "starting with the root node", which is misleading as the root node will never be returned. As such, we change the wording to "starting from the root node", in all of these iterators. This change includes the subtree iterators, as they could be passed a cursor to the root node.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0200**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **A.19**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0200 New types in Ada.Locales can't be converted to/from strings

Working Reference Number AI12-0037-1

Question

It is difficult to output objects of type Ada.Locales.Language_Code or Ada.Locales.Country_Code; you can't give them to Text_IO.Put because their type isn't String, but you also can't convert them to String because they run afoul of 4.6(24.5/2) which requires that component subtypes must statically match.

Should some change be made in order to make this easier? (Yes.)

Summary of Response

Types Ada.Locales.Language_Code and Ada.Locales.Country_Code should be derived from type String.

Corrigendum Wording

Replace A.19(4):

```
type Language_Code is array (1 .. 3) of Character range 'a' .. 'z';
type Country_Code is array (1 .. 2) of Character range 'A' .. 'Z';
```

by:

```
type Language_Code is new String (1 .. 3)
   with Dynamic_Predicate =>
     (for all E of Language_Code => E in 'a' .. 'z');
type Country_Code is new String (1 .. 2)
   with Dynamic_Predicate =>
     (for all E of Country_Code  => E in 'A' .. 'Z');
```

Discussion

The use of a predicate rather than constrained component subtypes means that this change is inconsistent with the original Ada 2012 definition - the exception raised on failure is changed. We don't consider that important as it is unlikely that users will construct these values and when they do, it is highly unlikely that they would handle any resulting exception (which would represent a bug).

We could have used a raise_expression as described in AI12-0022-1 to avoid this inconsistency, but the exception raised wasn't considered important enough to use the extra machinery. (We also avoid the problem addressed by AI12-0054-1 by using this formulation; any technique for raising Constraint_Error would suffer from that problem.)

The predicate could have been written in a number of other ways; this one doesn't depend explicitly on the bounds of the subtypes, so it is preferred. Of course, an implementation can replace this by anything equivalent if that will provide better performance.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0201**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **B.1; B.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0201 Enumeration types should be eligible for convention C

Working Reference Number AI12-0135-1

Question

It is convenient to map C enumeration types to Ada enumeration types. However, enumeration types are not eligible for convention C. An implementation can allow enumeration types to have convention C via B.1(20), but this depends on the implementer, and thus impacts portability. Should Ada provide this capability? (Yes.)

Summary of Response

Enumeration types are eligible for all languages that interface with Ada, subject to a limitation on their range of internal codes. More detailed requirements exist for mapping Ada enumeration types to C/C++ enum types.

Corrigendum Wording

Insert after B.1(14):

- Convention *L* has been specified for T, and T is *eligible for convention L*; that is:

the new paragraph:

- T is an enumeration type such that all internal codes (whether assigned by default or explicitly) are within an implementation-defined range that includes at least the range of values 0 .. 2**15–1;

Replace B.1(41):

For each supported convention *L* other than Intrinsic, an implementation should support specifying the Import and Export aspects for objects of *L*-compatible types and for subprograms, and the Convention aspect for *L*-eligible types and for subprograms, presuming the other language has corresponding features. Specifying the Convention aspect need not be supported for scalar types.

by:

For each supported convention *L* other than Intrinsic, an implementation should support specifying the Import and Export aspects for objects of *L*-compatible types and for subprograms, and the Convention aspect for *L*-eligible types and for subprograms, presuming the other language has corresponding features. Specifying the Convention aspect need not be supported for scalar types, other than enumeration types whose internal codes fall within the range 0 .. 2**15–1.

Insert after B.3(65):

- An Ada function corresponds to a non-void C function.

the new paragraph:

- An Ada enumeration type corresponds to a C enumeration type with corresponding enumeration literals having the same internal codes, provided the internal codes fall within the range of the C int type.

Response

Suggest support for interfacing with Ada enumeration types, so long as the number of internal codes is within some implementation defined range. Impose more detailed requirements for C/C++. (That is, include it in the Implementation Advice at the end of B.1.)

Discussion

For some reason, B.1(12-21/3) do not require any scalar types to be eligible for convention *L*. Perhaps that is because using the scalar types in the various interface packages is preferred.

That's not sensible in the case of enumeration types, however. Many languages have some type similar to Ada's enumerations, and direct mapping should be supported if possible. Therefore we have chosen to require support for enumeration types in all languages that interface with Ada, subject to an implementation-defined limitation on the range of internal codes.

We set the minimum supported range to be the non-negative values of the 16-bit signed integer type, since that is known to be supportable by C compilers, and is presumed to be supportable by any other language which supports integer types.

For C, we require that an Ada enumeration type maps to a C enumeration type, provided the internal codes fall within the range of "int", as that is the C requirement. C++ is more flexible, but there seems no reason to specify additional requirements for C++ interfacing.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0202**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **B.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0202 Import of variadic C functions

Working Reference Number AI12-0028-1

Question

B.3(75) contains the following statement: "10 A C function that takes a variable number of arguments can correspond to several Ada subprograms, taking various specific numbers and types of parameters.".

This is only true for ABIs where the regular C calling convention is the same as the calling convention used for variadic C functions. On systems using the System V ABI AMD64 (used e.g. on 64bit Linux, see http://www.x86-64.org/documentation/abi.pdf) the calling conventions are different, which leads to unpredictable results when importing variadic C functions.

Should this note be rewritten or deleted? (Yes.)

Should Ada have some standard way of interfacing to variadic C functions? (Yes.)

Summary of Response

A new convention, C_Variadic_*N* is introduced to handle C functions with variable numbers of parameters.

Corrigendum Wording

Replace B.3(1):

> The facilities relevant to interfacing with the C language and the corresponding subset of the C++ language are the package Interfaces.C and its children, and support for specifying the Convention aspect with *convention*_identifiers C and C_Pass_By_Copy.

by:

> The facilities relevant to interfacing with the C language and the corresponding subset of the C++ language are the package Interfaces.C and its children, and support for specifying the Convention aspect with *convention*_identifiers C, C_Pass_By_Copy, and any of the C_Variadic_*n* conventions described below.

Insert after B.3(60.15):

> If a type is C_Pass_By_Copy-compatible, then it is also C-compatible.

the new paragraph:

> The identifiers C_Variadic_0, C_Variadic_1, C_Variadic_2, and so on are *convention*_identifiers. These conventions are said to be *C_Variadic*. The convention C_Variadic_*n* is the calling convention for a variadic C function taking *n* fixed parameters and then a variable number of additional parameters. The C_Variadic_*n* convention shall only be specified as the convention aspect for a subprogram, or for an access-to-subprogram type, having at least *n* parameters. A type is compatible with a C_Variadic convention if and only if the type is C-compatible.

Replace B.3(75):

> A C function that takes a variable number of arguments can correspond to several Ada subprograms, taking various specific numbers and types of parameters.

by:

> A variadic C function can correspond to several Ada subprograms, taking various specific numbers and types of parameters.

Discussion

This AI is classified as a Binding Interpretation, applying to Ada 2012. Note that adding a language name is something that any implementation can do, so Ada 95 and Ada 2005 compilers also can implement the solution. As such the classification does not matter that much, but it seems better to encourage adoption in

Ada 2005 compilers (else it is impossible to write portable interfaces without resorting to writing C code, which should never be required lest programmers wonder why they're using Ada at all).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0203**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **C.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0203 Missing rules for Discard_Names aspect

Working Reference Number AI12-0072-1

Question

Discard_Names is not on the list of language-defined aspects and nothing in C.5 says that it can be used as an aspect name, but AARM Note C.5(6.a/3) says it is a language-defined aspect. Which is it? (It's an aspect.)

Summary of Response

Discard_Names is an aspect, with typical rules.

Corrigendum Wording

Replace the title of C.5:

Pragma Discard_Names

by:

Aspect Discard_Names

Replace C.5(1):

A pragma Discard_Names may be used to request a reduction in storage used for the names of certain entities.

by:

Specifying the aspect Discard_Names can be used to request a reduction in storage used for the names of entities with runtime name text.

Static Semantics

An entity with *runtime name text* is a nonderived enumeration first subtype, a tagged first subtype, or an exception.

For an entity with runtime name text, the following language-defined representation aspect may be specified:

Discard_Names
: The type of aspect Discard_Names is Boolean. If directly specified, the aspect_definition shall be a static expression. If not specified (including by inheritance), the aspect is False.

Replace C.5(5):

The local_name (if present) shall denote a nonderived enumeration [first] subtype, a tagged [first] subtype, or an exception. The pragma applies to the type or exception. Without a local_name, the pragma applies to all such entities declared after the pragma, within the same declarative region. Alternatively, the pragma can be used as a configuration pragma. If the pragma applies to a type, then it applies also to all descendants of the type.

by:

The local_name (if present) shall denote an entity with runtime name text. The pragma specifies that the aspect Discard_Names for the type or exception has the value True. Without a local_name, the pragma specifies that all entities with runtime name text declared after the pragma, within the same declarative region have the value True for aspect Discard_Names. Alternatively, the pragma can be used as a configuration pragma. If the configuration pragma Discard_Names applies to a compilation unit, all entities with runtime name text declared in the compilation unit have the value True for the aspect Discard_Names.

Replace C.5(7):

If the pragma applies to an enumeration type, then the semantics of the Wide_Wide_Image and Wide_Wide_Value attributes are implementation defined for that type; the semantics of Image,

Wide_Image, Value, and Wide_Value are still defined in terms of Wide_Wide_Image and Wide_Wide_Value. In addition, the semantics of Text_IO.Enumeration_IO are implementation defined. If the pragma applies to a tagged type, then the semantics of the Tags.Wide_Wide_Expanded_Name function are implementation defined for that type; the semantics of Tags.Expanded_Name and Tags.Wide_Expanded_Name are still defined in terms of Tags.Wide_Wide_Expanded_Name. If the pragma applies to an exception, then the semantics of the Exceptions.Wide_Wide_Exception_Name function are implementation defined for that exception; the semantics of Exceptions.Exception_Name and Exceptions.Wide_Exception_Name are still defined in terms of Exceptions.Wide_Wide_Exception_Name.

by:

If the aspect Discard_Names is True for an enumeration type, then the semantics of the Wide_Wide_Image and Wide_Wide_Value attributes are implementation defined for that type; the semantics of Image, Wide_Image, Value, and Wide_Value are still defined in terms of Wide_Wide_Image and Wide_Wide_Value. In addition, the semantics of Text_IO.Enumeration_IO are implementation defined. If the aspect Discard_Names is True for a tagged type, then the semantics of the Tags.Wide_Wide_Expanded_Name function are implementation defined for that type; the semantics of Tags.Expanded_Name and Tags.Wide_Expanded_Name are still defined in terms of Tags.Wide_Wide_Expanded_Name. If the aspect Discard_Names is True for an exception, then the semantics of the Exceptions.Wide_Wide_Exception_Name function are implementation defined for that exception; the semantics of Exceptions.Exception_Name and Exceptions.Wide_Exception_Name are still defined in terms of Exceptions.Wide_Wide_Exception_Name.

Replace C.5(8):

If the pragma applies to an entity, then the implementation should reduce the amount of storage used for storing names associated with that entity.

by:

If the aspect Discard_Names is True for an entity, then the implementation should reduce the amount of storage used for storing names associated with that entity.

Response

(See !summary.)

Discussion

The original intent was that blanket rules in 13.1 and 13.1.1 (especially 13.1.1(32/3)) would be sufficient to specify the rules for aspects associated with representation pragmas. But eventually every such pragma except this one got explicit rules. It seems weird that this solitary pragma depends on the blanket rules (enough that it failed to get added to the index of aspects).

In addition, it's not clear that the blanket rules explain what it means to specify the aspect using an aspect_specification. In particular, do the rules make the specifying the aspect to True with an aspect_specification the same as if the associated pragma applies to the entity? We sidestepped that for all other pragmas by defining the pragmas in terms of the aspects.

Finally, if we add the aspect to the aspect index, it would be bad if there was no text describing the aspect at the target of that index entry.

So, for all of these reasons, we define the aspect here, and then define the pragma in terms of the aspect in the usual way.

In doing this, an old bug was discovered, in that a configuration pragma is defined to apply to a compilation unit, not the entities defined in that unit (which is what we want here). That was corrected in these wording changes.

It is not the intent of these changes to change the effect of pragma Discard_Names (as understood, as opposed to formally defined) in any way.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0204**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Clarification requested**
REFERENCES IN DOCUMENT: **D.1; D.16**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0204 Real-time aspects need to specify when they are evaluated

Working Reference Number AI12-0081-1

Question

The definition of the various task aspects say that they are "evaluated for each task object". This does not say when that evaluation occurs. When are these aspects evaluated? (When the task object is created.)

Summary of Response

Real-time aspects of a type are evaluated when an object of the task type is created.

Corrigendum Wording

Replace D.1(17):

> The expression specified for the Priority or Interrupt_Priority aspect of a task is evaluated for each task object (see 9.1). For the Priority aspect, the value of the expression is converted to the subtype Priority; for the Interrupt_Priority aspect, this value is converted to the subtype Any_Priority. The priority value is then associated with the task object whose task declaration specifies the aspect.

by:

> The expression specified for the Priority or Interrupt_Priority aspect of a task type is evaluated each time an object of the task type is created (see 9.1). For the Priority aspect, the value of the expression is converted to the subtype Priority; for the Interrupt_Priority aspect, this value is converted to the subtype Any_Priority. The priority value is then associated with the task object.

Replace D.16(9):

> The expression specified for the CPU aspect of a task is evaluated for each task object (see 9.1). The CPU value is then associated with the task object whose task declaration specifies the aspect.

by:

> The expression specified for the CPU aspect of a task type is evaluated each time an object of the task type is created (see 9.1). The CPU value is then associated with the task object.

Discussion

The wording for aspect Dispatching_Domain was updated in AI12-0033-1.

Aspect Relative_Deadline already specifies when it is to be evaluated, so no wording change is needed for that aspect.

We don't need to change the wording of D.1(18/3), since the priority of the main subprogram has to be static, and thus does not need to be evaluated.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0205**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **D.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0205 The Priority aspect can be specified when Attach_Handler is specified

Working Reference Number AI12-0051-1

Question

Consider:

```
protected Monitor is
   with Priority => System.Priority'Last;

   procedure Handler
      with Attach_Handler => ...;

   entry Wait;
private
   Signaled : Boolean := False;
end Monitor;
```

What is the initial priority of Monitor? D.3(6.1/3) says that it is System.Priority'Last, while D.3(10/3) says that it is some implementation-defined value (because Interrupt_Priority isn't specified). Which is right? (D.3(6.1/3)).

Summary of Response

The Priority aspect can specify the priority of an interrupt handler.

Corrigendum Wording

Replace D.3(10):

- If an Interrupt_Handler or Attach_Handler aspect (see C.3.1) is specified for a protected subprogram of a protected type that does not have the Interrupt_Priority aspect specified, the initial priority of protected objects of that type is implementation defined, but in the range of the subtype System.Interrupt_Priority.

by:

- If an Interrupt_Handler or Attach_Handler aspect (see C.3.1) is specified for a protected subprogram of a protected type that does not have either the Priority or Interrupt_Priority aspect specified, the initial priority of protected objects of that type is implementation defined, but in the range of the subtype System.Interrupt_Priority.

Discussion

It's clear that either D.3(6.1/3) or D.3(10/3) is wrong. C.3.1(11/3) specifies a runtime check for the priority value, which would be unnecessary if only Interrupt_Priority was allowed. Moreover, if an implementation allowed Priority and made the runtime check, it would be incompatible to make specifying the Priority aspect illegal (which would be especially annoying if the given priority was in the appropriate range). Liberalizing D.3(10/3) has no such compatibility problem, so that is the fix we make.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0206**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **D.7; D.13**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0206 All properties of a usage profile are defined by pragmas

Working Reference Number AI12-0055-1

Question

The definition of a profile says (13.12(13/3)):

A profile is equivalent to the set of configuration pragmas that is defined for each usage profile.

However, the Ravenscar profile has rules (D.13.1(8-9/3)) that are not associated with any configuration pragma.

Should one be defined? (Yes.)

Summary of Response

The application of the Ravenscar Profile to multiprocessors requires one new configuration pragma and one new restriction to be defined to ensure that all tasks are assigned to a CPU.

Corrigendum Wording

Insert after D.7(10):

No_Dynamic_Attachment
: There is no use of a name denoting any of the operations defined in package Interrupts (Is_Reserved, Is_Attached, Current_Handler, Attach_Handler, Exchange_Handler, Detach_Handler, and Reference).

the new paragraph:

No_Dynamic_CPU_Assignment
: No task has the CPU aspect specified to be a non-static expression. Each task (including the environment task) that has the CPU aspect specified as Not_A_Specific_CPU will be assigned to a particular implementation-defined CPU. The same is true for the environment task when the CPU aspect is not specified. Any other task without a CPU aspect will activate and execute on the same processor as its activating task.

Replace D.13(6):

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
pragma Detect_Blocking;
pragma Restrictions (
                No_Abort_Statements,
                No_Dynamic_Attachment,
                No_Dynamic_Priorities,
                No_Implicit_Heap_Allocations,
                No_Local_Protected_Objects,
                No_Local_Timing_Events,
                No_Protected_Type_Allocators,
                No_Relative_Delay,
                No_Requeue_Statements,
                No_Select_Statements,
                No_Specific_Termination_Handlers,
                No_Task_Allocators,
                No_Task_Hierarchy,
                No_Task_Termination,
                Simple_Barriers,
                Max_Entry_Queue_Length => 1,
                Max_Protected_Entries => 1,
                Max_Task_Entries => 0,
                No_Dependence => Ada.Asynchronous_Task_Control,
                No_Dependence => Ada.Calendar,
```

```
                    No_Dependence => Ada.Execution_Time.Group_Budgets,
                    No_Dependence => Ada.Execution_Time.Timers,
                    No_Dependence => Ada.Task_Attributes,
                    No_Dependence =>
System.Multiprocessors.Dispatching_Domains);
```

by:

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
pragma Detect_Blocking;
pragma Restrictions (
                    No_Abort_Statements,
                    No_Dynamic_Attachment,
                    No_Dynamic_CPU_Assignment,
                    No_Dynamic_Priorities,
                    No_Implicit_Heap_Allocations,
                    No_Local_Protected_Objects,
                    No_Local_Timing_Events,
                    No_Protected_Type_Allocators,
                    No_Relative_Delay,
                    No_Requeue_Statements,
                    No_Select_Statements,
                    No_Specific_Termination_Handlers,
                    No_Task_Allocators,
                    No_Task_Hierarchy,
                    No_Task_Termination,
                    Simple_Barriers,
                    Max_Entry_Queue_Length => 1,
                    Max_Protected_Entries => 1,
                    Max_Task_Entries => 0,
                    No_Dependence => Ada.Asynchronous_Task_Control,
                    No_Dependence => Ada.Calendar,
                    No_Dependence => Ada.Execution_Time.Group_Budgets,
                    No_Dependence => Ada.Execution_Time.Timers,
                    No_Dependence => Ada.Task_Attributes,
                    No_Dependence =>
System.Multiprocessors.Dispatching_Domains);
```

Delete D.13(8):

A task shall only be on the ready queues of one processor, and the processor to which a task belongs shall be defined statically. Whenever a task running on a processor reaches a task dispatching point, it goes back to the ready queues of the same processor. A task with a CPU value of Not_A_Specific_CPU will execute on an implementation defined processor. A task without a CPU aspect will activate and execute on the same processor as its activating task.

Insert after D.13(10):

NOTES

42 The effect of the Max_Entry_Queue_Length => 1 restriction applies only to protected entry queues due to the accompanying restriction of Max_Task_Entries => 0.

the new paragraphs:

43 When the Ravenscar profile is in effect (via the effect of the No_Dynamic_CPU_Assignment restriction), all of the tasks in the partition will execute on a single CPU unless the programmer explicitly uses aspect CPU to specify the CPU assignments for tasks. The use of multiple CPUs requires care, as many guarantees of single CPU scheduling no longer apply.

44 It is not recommended to specify the CPU of a task to be Not_A_Specific_CPU when the Ravenscar profile is in effect. How a partition executes strongly depends on the assignment of tasks to CPUs.

Response

(See !summary.)

Discussion

During the discussion of AI12-0048-1, it was pointed out that Ravenscar has special rules for the assignment of tasks to processors. These rules appear to require that FIFO_within_Priorities scheduling be ineffective on Ravenscar programs (as it appears impossible in general for the implementation to assign tasks such that

unbounded priority inversion cannot occur]). [That is possible for some systems of tasks and priorities, and it can often be done by hand, but the rules require the system to do it automatically.] It was felt by some that hiding such a significant change to the priority model in an Implementation Requirement is far too subtle.

At this point, it was noted that Ravenscar is a profile, which is just a collection of pragmas. But the behavior of D.13(8-9/3) is not tied to any pragma. This is wrong. Moreover, by defining it as a pragma, both the rules and the effects would be more visible.

To fix this we add a new restriction No_Dynamic_CPU_Assignment.

We add this restriction to the Ravenscar profile.

D.13(8/3) is deleted; it follows from other rules. The text about "ready queues" is obvious (it's the definition of assigning a CPU to a task); and the Not_A_Specific_CPU text is moved to the new restriction, along with the redundant text.

We also add user notes to the Ravenscar profile to note that a Ravenscar program will always run on a single CPU unless the programmer takes action with aspect CPU to define otherwise. We also caution against specifying tasks to have Not_A_Specific_CPU.

It's possible that this new restriction is incompatible with existing implementations. However, that could only be because they ignored D.13(8/3)'s requirement that "the processor be defined statically"; specifying a CPU aspect with a non-static value could not meet that requirement.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0207**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **D.7**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0207 Restriction No_Tasks_Unassigned_To_CPU

Working Reference Number AI12-0117-1

Question

The Ravenscar Profile on multiprocessors requires that all tasks are assigned to CPUs. In order to do this safely, we need a new restriction to be defined to ensure that every task is assigned to a known, particular CPU.

We need to know the CPU of each task so that it is possible to analyze the scheduling of each task, since each CPU has a separate ready queue. This means that two tasks assigned to the same CPU will behave differently than the same two tasks assigned to different CPUs.

Should we define such a restriction? (Yes.)

Summary of Response

Add restriction No_Tasks_Unassigned_To_CPU to provide safe use of Ravenscar.

Corrigendum Wording

Insert after D.7(10.8):

No_Specific_Termination_Handlers
: There is no use of a name denoting the Set_Specific_Handler and Specific_Handler subprograms in Task_Termination.

the new paragraph:

No_Tasks_Unassigned_To_CPU
: The CPU aspect is specified for the environment task. No CPU aspect is specified to be statically equal to Not_A_Specific_CPU. If aspect CPU is specified (dynamically) to the value Not_A_Specific_CPU, then Program_Error is raised. If Set_CPU or Delay_Until_And_Set_CPU are called with the CPU parameter equal to Not_A_Specific_CPU, then Program_Error is raised.

Discussion

We add a new restriction:

No_Tasks_Unassigned_To_CPU

to signify that (1) no task is assigned a CPU value of Not_A_Specific_CPU, and (2) the environment task has a CPU specified.

This ensures that all tasks are assigned to a known, particular CPU. (If a task does not have a specified CPU, it will run on the CPU of the task that activated it -- that is the standard semantics for CPU assignment. Thus we only have to ensure that the environment task has a specified CPU and that no task is explicitly assigned Not_A_Specific_CPU.)

The purpose of this restriction is to ensure that there is no implementation-defined assignment of CPUs; that ensures that the scheduling of tasks can be analyzed.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0208**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **D.13**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0208 Synchronous Barriers are not allowed with Ravenscar

Working Reference Number AI12-0073-1

Question

Ada.Synchronous_Barriers allows for several different implementations. It could be directly implemented with target-provided library routines, such as are provided by POSIX. Alternatively, the package could be completely written in Ada using a protected object approach.

However, a protected object approach would make the barrier objects unusable (other than at library-level), as the restriction No_Local_Protected_Objects applies when Ravenscar is specified. This poses a portability problem.

Should this portability problem be addressed somehow? (Yes.)

Summary of Response

Synchronous Barriers are not allowed when profile Ravenscar is in effect.

Corrigendum Wording

Replace D.13(6):

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
pragma Detect_Blocking;
pragma Restrictions (
                No_Abort_Statements,
                No_Dynamic_Attachment,
                No_Dynamic_Priorities,
                No_Implicit_Heap_Allocations,
                No_Local_Protected_Objects,
                No_Local_Timing_Events,
                No_Protected_Type_Allocators,
                No_Relative_Delay,
                No_Requeue_Statements,
                No_Select_Statements,
                No_Specific_Termination_Handlers,
                No_Task_Allocators,
                No_Task_Hierarchy,
                No_Task_Termination,
                Simple_Barriers,
                Max_Entry_Queue_Length => 1,
                Max_Protected_Entries => 1,
                Max_Task_Entries => 0,
                No_Dependence => Ada.Asynchronous_Task_Control,
                No_Dependence => Ada.Calendar,
                No_Dependence => Ada.Execution_Time.Group_Budgets,
                No_Dependence => Ada.Execution_Time.Timers,
                No_Dependence => Ada.Task_Attributes,
                No_Dependence =>
System.Multiprocessors.Dispatching_Domains);
```

by:

```
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
pragma Detect_Blocking;
pragma Restrictions (
                No_Abort_Statements,
                No_Dynamic_Attachment,
                No_Dynamic_Priorities,
                No_Implicit_Heap_Allocations,
                No_Local_Protected_Objects,
                No_Local_Timing_Events,
```

```
                    No_Protected_Type_Allocators,
                    No_Relative_Delay,
                    No_Requeue_Statements,
                    No_Select_Statements,
                    No_Specific_Termination_Handlers,
                    No_Task_Allocators,
                    No_Task_Hierarchy,
                    No_Task_Termination,
                    Simple_Barriers,
                    Max_Entry_Queue_Length => 1,
                    Max_Protected_Entries => 1,
                    Max_Task_Entries => 0,
                    No_Dependence => Ada.Asynchronous_Task_Control,
                    No_Dependence => Ada.Calendar,
                    No_Dependence => Ada.Execution_Time.Group_Budgets,
                    No_Dependence => Ada.Execution_Time.Timers,
                    No_Dependence => Ada.Synchronous_Barriers,
                    No_Dependence => Ada.Task_Attributes,
                    No_Dependence =>
System.Multiprocessors.Dispatching_Domains);
```

Response

(See !summary.)

Discussion

Ravenscar is defined in order to support a simple Ada runtime which provides behavior that can be analyzed (both of the user programs and of the runtime itself). One of the major simplifications is "no queueing" -- that is, no more than one task can be blocked at any point in the program. The purpose of synchronous barriers is to block multiple tasks and allow simultaneous release; as such it is incompatible with the existing Ravenscar rules.

This point was missed by the questioner; if a protected type implementation was used, the maximum number of tasks that could use a synchronous barrier in a Ravenscar program would be 1, which would defeat the purpose of the synchronous barrier feature.

It has been argued that analysis of synchronous barriers is easier than that of regular queuing, because the tasks are all released at the same time. Even if that is true, it is only part of the picture, as the increased complication of the Ravenscar runtime remains. In particular, the runtime would now require a ready queue, which is not required for the existing Ravenscar runtimes (assuming a special mechanism for task activation). Adding such a complication would surely make Ravenscar runtimes harder to analyze and verify.

Historically, we have been very reluctant to add new complications to the Ravenscar profile (for instance, the request of AI05-0172-1 was rejected). As such, we do not allow this new facility to be used in Ravenscar programs. Note that this is the most flexible option for the future, as we could compatibly adopt some other solution should it prove necessary. Other options do not provide this flexiblity.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0209**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **D.16.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0209 Sets of CPUs when defining dispatching domains

Working Reference Number AI12-0033-1

Question

It seems limiting that dispatching domains can only be defined by ranges of CPUs.

For example in our hardware Architecture, we have 4 CPU's with each 4 cores, but the numbering is not straight forward. The first CPU has the cores 0, 4, 8, 12, and the second 1,5,9,13. So here it will be better to have a higher flexibility in assigning Domains to ranges and single cores e.g a Domain (2,6,10,14) pointing the 2nd CPU or (2-3,6-7,10-11,14-15) defining a Domain of the 2nd and 3rd CPU.

Should an additional Create routine be defined? (Yes.)

Summary of Response

Discontiguous sets of CPU numbers may be used when specifying a dispatching domain. A dispatching domain may be empty, but it is an error to assign a task to an empty domain.

Corrigendum Wording

Replace D.16.1(7):

```
function Create (First, Last : CPU) return Dispatching_Domain;
```

by:

```
function Create (First : CPU; Last : CPU_Range) return
Dispatching_Domain;
```

Replace D.16.1(9):

```
function Get_Last_CPU  (Domain : Dispatching_Domain) return CPU;
```

by:

```
function Get_Last_CPU  (Domain : Dispatching_Domain) return CPU_Range;
type CPU_Set is array(CPU range <>) of Boolean;
function Create (Set : CPU_Set) return Dispatching_Domain;
function Get_CPU_Set (Domain : Dispatching_Domain) return CPU_Set;
```

Replace D.16.1(20):

> The expression specified for the Dispatching_Domain aspect of a task is evaluated for each task object (see 9.1). The Dispatching_Domain value is then associated with the task object whose task declaration specifies the aspect.

by:

> The expression specified for the Dispatching_Domain aspect of a task type is evaluated each time an object of the task type is created (see 9.1). If the identified dispatching domain is empty, then Dispatching_Domain_Error is raised; otherwise the newly created task is assigned to the domain identified by the value of the expression.

Replace D.16.1(23):

> The function Create creates and returns a Dispatching_Domain containing all the processors in the range First .. Last. These processors are removed from System_Dispatching_Domain. A call of Create will raise Dispatching_Domain_Error if any designated processor is not currently in System_Dispatching_Domain, or if the system cannot support a distinct domain over the processors identified, or if a processor has a task assigned to it, or if the allocation would leave System_Dispatching_Domain empty. A call of Create will raise Dispatching_Domain_Error if the calling task is not the environment task, or if Create is called after the call to the main subprogram.

by:

The function Create with First and Last parameters creates and returns a dispatching domain containing all the processors in the range First .. Last. The function Create with a Set parameter creates and returns a dispatching domain containing the processors for which Set(I) is True. These processors are removed from System_Dispatching_Domain. A call of Create will raise Dispatching_Domain_Error if any designated processor is not currently in System_Dispatching_Domain, or if the system cannot support a distinct domain over the processors identified, or if a processor has a task assigned to it, or if the allocation would leave System_Dispatching_Domain empty. A call of Create will raise Dispatching_Domain_Error if the calling task is not the environment task, or if Create is called after the call to the main subprogram.

Replace D.16.1(24):

The function Get_First_CPU returns the first CPU in Domain; Get_Last_CPU returns the last one.

by:

The function Get_First_CPU returns the first CPU in Domain, or CPU'First if Domain is empty; Get_Last_CPU returns the last CPU in Domain, or CPU_Range'First if Domain is empty. The function Get_CPU_Set(D) returns an array whose low bound is Get_First_CPU(D), whose high bound is Get_Last_CPU(D), with True values in the Set corresponding to the CPUs that are in the given Domain.

Replace D.16.1(26):

A call of the procedure Assign_Task assigns task T to the CPU within Dispatching_Domain Domain. Task T can now execute only on CPU unless CPU designates Not_A_Specific_CPU, in which case it can execute on any processor within Domain. The exception Dispatching_Domain_Error is propagated if T is already assigned to a Dispatching_Domain other than System_Dispatching_Domain, or if CPU is not one of the processors of Domain (and is not Not_A_Specific_CPU). A call of Assign_Task is a task dispatching point for task T unless T is inside of a protected action, in which case the effect on task T is delayed until its next task dispatching point. If T is the Current_Task the effect is immediate if T is not inside a protected action, otherwise the effect is as soon as practical. Assigning a task to System_Dispatching_Domain that is already assigned to that domain has no effect.

by:

A call of the procedure Assign_Task assigns task T to the CPU within the dispatching domain Domain. Task T can now execute only on CPU, unless CPU designates Not_A_Specific_CPU in which case it can execute on any processor within Domain. The exception Dispatching_Domain_Error is propagated if Domain is empty, T is already assigned to a dispatching domain other than System_Dispatching_Domain, or if CPU is not one of the processors of Domain (and is not Not_A_Specific_CPU). A call of Assign_Task is a task dispatching point for task T unless T is inside of a protected action, in which case the effect on task T is delayed until its next task dispatching point. If T is the Current_Task the effect is immediate if T is not inside a protected action, otherwise the effect is as soon as practical. Assigning a task already assigned to System_Dispatching_Domain to that domain has no effect.

Response

A more flexible specification is proposed allowing sets of CPUs to be specified. This may in any case be necessary to describe the set of CPUs that remain in the System dispatching domain, after the other domains have been "carved" out of it.

Discussion

There is already a problem with the current mechanism for specifying dispatching domains, in that the System_Dispatching_Domain might have holes. For example, if the program is running on a system with 8 CPUs, and a domain is Created containing CPUs 3 through 6, the System_Dispatching_Domain holds the remaining CPUs, which in this case are CPUs 1, 2, 7, and 8. Then Get_First_CPU(System_Dispatching_Domain) = 1 and Get_First_CPU(System_Dispatching_Domain) = 8. but this surely does not completely characterize the values in the System_Dispatching_Domain. Moreover, you can create many holes in the System_Dispatching_Domain this way.

So we propose to add set operations here in order to properly represent any of these items.

We have chosen a straightforward bit-vector representation of the CPU set. A more sophisticated approach is possible, but seems to be overkill. The CPU_Set array type is unconstrained, so the overall length may be kept to the minimum necessary to include all of the "True" values. Aspect Pack might be applied to CPU_Set in the private part of this package, if the implementation so chooses.

Note that if the domain specified by Create is empty, Get_Last_CPU(D) will return zero and Get_First_CPU(D) will return one. We allow empty domains to be specified because the domains might be created using information from querying the environment, and in some environments there might be insufficient CPUs to make each possible domain non-empty. The code which assigns tasks can be conditional, but it is not easy to make the declarations of dispatching domains conditional, as they must be declared and initialized using library-level declarations.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0210**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **D.16.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0210 Definition of "dispatching domain"

Working Reference Number AI12-0082-1

Question

There are many occurrences where Dispatching_Domain (the name of a type) in D.16.1 should be replaced by "dispatching domain" (the semantic concept behind the type) or some similar change. For instance, one does not assign to Dispatching_Domain, as it is a type. Fix this? (Yes.)

Summary of Response

"Dispatching_Domain" is replaced by "dispatching domain" in most uses, and "dispatching domain" is defined appropriately.

Corrigendum Wording

Replace D.16.1(16):

> The type Dispatching_Domain represents a series of processors on which a task may execute. Each processor is contained within exactly one Dispatching_Domain. System_Dispatching_Domain contains the processor or processors on which the environment task executes. At program start-up all processors are contained within System_Dispatching_Domain.

by:

> A *dispatching domain* represents a set of processors on which a task may execute. Each processor is contained within exactly one dispatching domain. An object of type Dispatching_Domain identifies a dispatching domain. System_Dispatching_Domain identifies a domain that contains the processor or processors on which the environment task executes. At program start-up all processors are contained within this domain.

Replace D.16.1(22):

> If both Dispatching_Domain and CPU are specified for a task, and the CPU value is not contained within the range of processors for the domain (and is not Not_A_Specific_CPU), the activation of the task is defined to have failed, and it becomes a completed task (see 9.2).

by:

> If both the dispatching domain and CPU are specified for a task, and the CPU value is not contained within the set of processors for the domain (and is not Not_A_Specific_CPU), the activation of the task is defined to have failed, and it becomes a completed task (see 9.2).

Replace D.16.1(25):

> The function Get_Dispatching_Domain returns the Dispatching_Domain on which the task is assigned.

by:

> The function Get_Dispatching_Domain returns the dispatching domain on which the task is assigned.

Replace D.16.1(27):

> A call of procedure Set_CPU assigns task T to the CPU. Task T can now execute only on CPU, unless CPU designates Not_A_Specific_CPU, in which case it can execute on any processor within its Dispatching_Domain. The exception Dispatching_Domain_Error is propagated if CPU is not one of the processors of the Dispatching_Domain on which T is assigned (and is not Not_A_Specific_CPU). A call of Set_CPU is a task dispatching point for task T unless T is inside of a protected action, in which case the effect on task T is delayed until its next task dispatching point. If T is the Current_Task the effect is immediate if T is not inside a protected action, otherwise the effect is as soon as practical.

by:

> A call of procedure Set_CPU assigns task T to the CPU. Task T can now execute only on CPU, unless CPU designates Not_A_Specific_CPU, in which case it can execute on any processor within its

dispatching domain. The exception Dispatching_Domain_Error is propagated if CPU is not one of the processors of the dispatching domain on which T is assigned (and is not Not_A_Specific_CPU). A call of Set_CPU is a task dispatching point for task T unless T is inside of a protected action, in which case the effect on task T is delayed until its next task dispatching point. If T is the Current_Task the effect is immediate if T is not inside a protected action, otherwise the effect is as soon as practical.

Replace D.16.1(29):

A call of Delay_Until_And_Set_CPU delays the calling task for the designated time and then assigns the task to the specified processor when the delay expires. The exception Dispatching_Domain_Error is propagated if P is not one of the processors of the calling task's Dispatching_Domain (and is not Not_A_Specific_CPU).

by:

A call of Delay_Until_And_Set_CPU delays the calling task for the designated time and then assigns the task to the specified processor when the delay expires. The exception Dispatching_Domain_Error is propagated if P is not one of the processors of the calling task's dispatching domain (and is not Not_A_Specific_CPU).

Discussion

Unfortunately, just using "dispatching domain" brings up the question of what is a "dispatching domain" (as opposed to a "Dispatching_Domain"). The standard uses the term but never defines it.

Thus, the first thing we have to do is define the term "dispatching domain". The easiest solution is to define it to be "an object of type Dispatching_Domain". However, since a CPU can belong to only one dispatching domain, and Get_Dispatching_Domain returns what is effectively a reference to an existing domain, this doesn't work.

Thus we have to turn the definition around and make "an object of type Dispatching_Domain identify a dispatching domain", with "dispatching domain" defined similarly to "Type Dispatching_Domain" in the existing wording.

Some changes from "Dispatching_Domain" to "dispatching domain" were made in AI12-0033-1 (in paragraphs modified by that AI).

We change "series" (D.16.1(16/3)) and "range" (D.16.1(22/3)) to "set" as a dispatching domain might be a discontiguous set of CPU values and the words "series" and "range" seem to imply a continguous range of CPU values.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0211**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **D.16.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0211 Default behavior of tasks on a multiprocessor with a specified dispatching poli

Working Reference Number AI12-0048-1

Question

The language does not appear to say anything about which tasks are allowed to run on which CPUs of a multiprocessor in the absence of using any D.16 or D.16.1 facilities. Some real-time models map each task to a single CPU by default. This seems as if it could cause unbounded priority inversion.

For instance, consider an implementation which mapped all tasks of a particular task type to a single processor and the following state:

Two tasks of type T1 are ready at priorty 10

Two tasks of type T2 are ready at priority 6

All T1 tasks are mapped to processor 1

All T2 tasks are mapped to processor 2

```
with this mapping, you could have a ready priority 10 task not running and a
ready priority
6 task running. Presuming that this program specified a language-defined
dispatching policy,
this situation cannot be tolerated.
```

Should the wording be extended to cover this case? (Yes.)

Summary of Response

In the absence of any setting of the CPU of a task and the creation of any dispatching domains, a partition that specifies a language-defined dispatching policy will allow all tasks to run on all processors.

Corrigendum Wording

Insert after D.16.1(30):

> The implementation shall perform the operations Assign_Task, Set_CPU, Get_CPU and Delay_Until_And_Set_CPU atomically with respect to any of these operations on the same dispatching_domain, processor or task.

the new paragraph:

> Any task that belongs to the system dispatching domain can execute on any CPU within that domain, unless the assignment of the task has been specified.

Discussion

Annex D compliance is required when a dispatching policy is specified for the partition. In that case, there should only be a single dispatching domain (the system dispatching domain) that contains all of the processors on the systems; and all tasks should be eligible to run on all processors.

This is necessary so that priority inversion cannot occur in a program that is run on multiple processors but does not use any of the facilities in D.16. For instance, it must never be the case that a lower priority task is running while a higher priority task is ready if the dispatching policy is FIFO_within_Priorities. Automatically assigning tasks as described in the question is not allowed (the user can always make such assignment themselves).

Note that the default behavior of a program that does not specify a dispatching policy is implementation-defined; in particular, there is no requirement on how the tasks are mapped to processors in that case. (It is OK, for instance, for the tasks to be mapped one per processor in that case).

Also note that using Set_CPU for a task or assigning a task to a different dispatching domain eliminates any protection against priority inversion; we assume that users are aware of these possibilities and have taken steps to ensure that nothing bad happens in that case.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0212**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **E.2.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0212 Shared_Passive package restrictions

Working Reference Number AI12-0038-1

Question

Shared passive packages are allowed to depend on declared-pure packages. In Ada 2005, access type declarations were allowed in declared-pure packages, but the implications of this were not accounted for in the restrictions on shared passive packages. Should we add new restrictions directly on shared passive packages to account for this change? (Yes)

Summary of Response

Shared_Passive restrictions have to be adjusted because declared-pure packages now allow the declarations of access types.

Corrigendum Wording

Replace E.2.1(7):

- it shall not contain a library-level declaration of an access type that designates a class-wide type, task type, or protected type with entry_declarations.

by:

- it shall not contain a library-level declaration of an access type that designates a class-wide type, nor a type with a part that is of a task type or protected type with entry_declarations;
- it shall not contain a library-level declaration that contains a name that denotes a type declared within a declared-pure package, if that type has a part that is of an access type; for the purposes of this rule, the parts considered include those of the full views of any private types or private extensions.

Response

Shared passive packages shall not reference within a library-level declaration a type from a declared-pure package that contains a part that is of an access type. Note that this applies even if the type from the declared-pure package is a private type, and hence breaks the normal rules of privacy.

Discussion

The change to the existing bullet of E.2.1(7/1) is recognizing that tasks and protected objects with entries are bad news in an access collection (aka heap) within a shared-passive partition, whether they are whole objects or parts of other objects.

The new bullet following E.2.1(7/1) is intended to bolster E.2.1(8) to ensure that no access value of a "pure" access type could be embedded within an object of a shared-passive package, since it might point "back" into the data area of some particular active partition.

Shared passive packages are not allowed to declare access-to-class-wide types, but there is nothing preventing them from referencing an access-to-class-wide type declared in a declared-pure package. Furthermore, there is nothing preventing them using a type that has a part that is of an access type declared in a declared-pure package. The special shared-passive accessibility rules (E.2.1(8)) prevent creating values of an access type declared within the shared-passive package itself, that designate objects that might not be within the semantic closure of the shared-passive package. But these special accessibility rules don't apply to types declared in declared-pure packages, and hence values of such types might include references to such shorter-lived objects.

Example

Here is an example to help illustrate the purpose of the original E.2.1(8) rule:

```
package P1 is
   X : aliased Integer;
end P1;

package P2 is
   pragma Shared_Passive(P2);
   type Acc is access all Integer;
end P2;

with P1, P2;
package P3 is
   Z : P2.Acc := P1.X'access; -- legal? (No.)
end P3;
```

P2 does not depend semantically on P1, so the above is illegal. However, if we were to add a "with P1" in P2's context clause, then it would be legal, provided, of course, that P2 could legally depend on P1. That is only possible if P1 is itself a shared-passive package, since it cannot be "Pure" given that it has a variable. (Note that we could make X an aliased constant, and make "Acc" an "access constant" type, and then P1 could be a Pure package.)

This rule makes sense because unless P2 depends on P1, there is no guarantee that P1 will live as long as P2. We don't want a value of type P2.Acc designating an object in a package that doesn't live as long as P2. For shared-passive packages, it is semantic dependence that determines relative longevity.

Here is the problem that the new paragraph after E.2.1(7/1) is intended to address:

```
package P0 is
   pragma Pure(P0);
   type Trouble is private;
   function Make_Trouble(Z : access Integer) return Trouble;
private
   type Pure_Acc is access all Integer;
   for Pure_Acc'Storage_Size use 0;
   type Trouble is record
      PA : Pure_Acc;
   end record;

   function Make_Trouble(Z : access Integer) return Trouble
     is (Trouble'(PA => Pure_Acc(Z)));
end P0;

package P1 is
   X : aliased Integer;
end P1;

with P0;
package P2 is
   pragma Shared_Passive(P2);
   type Rec is record
      Troub : P0.Trouble;  --  legal? (No.)
   end record;
end P2;

with P1, P2;
package P3 is
   Z : P2.Rec := (Troub => P0.Make_Trouble(X'access));  -- not allowed
end P3;
```

Here we have used a type from the pure package P0 within a library-level type declaration in shared-passive package P2, and that type has a part that is of an access type. In P3 we set a value of that type to designate an object in P1. Just because P0 was originally declared in a pure package doesn't change the story. We definitely don't want any type defined in P2 to have a value that contains a pointer to an object in a package that doesn't live as long as P2. So with this new paragraph we simply disallow a shared-passive package using such a type, even though it requires us to break normal privacy rules.

We originally tried to augment the special accessibility rule, but that was inadequate, because accessibility of components is not re-checked on composite assignment, and so we felt we had to prevent the declaration or use of composite types in shared-passive packages that might carry pointers to shorter-lived objects.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0213**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **E.2.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0213 Remote stream attribute calls

Working Reference Number AI12-0034-1

Question

It should not be possible to make a remote stream attribute call (because such a stream attribute call by definition could not be used in marshalling and unmarshalling), and because all stream attributes contain an anonymous access type that does not support external streaming.

However, it is certainly possible to do this with a remote access-to-class-wide type. For instance:

```
with Ada.Streams;
package Pack1 is
    pragma Pure;
    type Root is tagged limited private;
    procedure Prim (N : Integer; X : Root);
    package Inner is   -- needed to make Do_Write non-primitive
        procedure Do_Write
                    (Stream : access Ada.Streams.Root_Stream_Type'Class;
                     X : Root);
    end Inner;
    for Root'Write use Inner.Do_Write;
private
    type Root is tagged limited record
        F1 : Integer;
    end record;
end Pack1;

with Pack1;
package Pack2 is
    pragma Remote_Types;
    type Root_Acc is access all Pack1.Root'Class;
end Pack2;

with Pack1, Pack2, Ada.Streams;
procedure Test1 (The_Obj : Pack2.Root_Acc) is
begin
    Pack1.Root'Class'Write (Stream, The_Obj.all);  -- Legal? (No.)
    Pack1.Inner.Do_Write (Stream, The_Obj.all); -- Illegal.
    Pack1.Prim (4, The_Obj.all); -- OK.
end Test1;
```

Should this be fixed? (Yes.)

Summary of Response

Dereferencing a remote access-to-class-wide type to make a dispatching call to a stream attribute is not allowed.

Corrigendum Wording

Replace E.2.2(16):

- A value of a remote access-to-class-wide type shall be dereferenced (or implicitly converted to an anonymous access type) only as part of a dispatching call where the value designates a controlling operand of the call (see E.4, "Remote Subprogram Calls");

by:

- A value of a remote access-to-class-wide type shall be dereferenced (or implicitly converted to an anonymous access type) only as part of a dispatching call to a primitive operation of the designated type where the value designates a controlling operand of the call (see E.4, "Remote Subprogram Calls");

Discussion

We could require the root designated type to not have available stream attributes. However, this solution is incompatible, and potentially penalizes users that do not use any problematic calls to stream attributes.

We can't ban the declarations of the subprograms like we do for other subprograms with access parameters, because that would make it impossible to define marshalling and unmarshalling for these types. Which also would make remote calls impossible, defeating the entire purpose.

So we modify E.2.2(16) to prevent this specific problem. This leaves the concern that the door is left open for other ways to cause the problem to be uncovered. It also complicates the model (there is nothing wrong with the dereference, given that it is used in a dispatching call); the problem is that the call has a profile that shouldn't be allowed. Blaming that on the access type usage seems wrong.

But this solution is compatible with all programs except those that actually try to do the bad thing. Thus it seems the best of a bad lot of solutions.

Note that there is another issue brought up by this example that we're not fixing.

Specifically, note that the calls are defined to be remote, even though there is no RCI packages in this partition (and no other partitions). That means that marshalling and unmarshalling and PCS use is required, even though the call has to be ultimately local. Even in programs that have multiple partitions, types declared in Pure and Remote_Types packages will be replicated in each partition, meaning that the associated calls on primitives are going to be local.

Moreover, it is necessary to dispatch in order to do marshalling (that's the only way to know how to marshall the specific type of the designated object), and it is silly to dispatch multiple times when once would do. Thus, it's likely that the implementation will marshall in stubs that are dispatched to, one for each specific type; with that implementation there is no need for remote calls (specifically marshalling and unmarshalling) if the specific subprogram is local. As such, there should be a permission to make local calls to local routines (and perhaps an aspect "All_Calls_Remote" on the remote access-to-class-wide to revoke that permission if that is actually useful).

[This second issue is related to the one covered by AI12-0031-1. That AI concludes that optimization of calls is in fact allowed unless All_Calls_Remote is given; see that AI.]

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0214**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **E.2.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0214 Missing aspect cases for Remote_Types

Working Reference Number AI12-0085-1

Question

E.2.2(17/2) talks about the Storage_Pool and Storage_Size attributes. It doesn't mention the corresponding aspects, which appears to allow specifying them. Should specifying them be disallowed as well? (Yes.)

Summary of Response

A remote access-to-class-wide type cannot specify the Storage_Pool or Storage_Size aspects.

Corrigendum Wording

Replace E.2.2(17):

- The Storage_Pool attribute is not defined for a remote access-to-class-wide type; the expected type for an allocator shall not be a remote access-to-class-wide type. A remote access-to-class-wide type shall not be an actual parameter for a generic formal access type. The Storage_Size attribute of a remote access-to-class-wide type yields 0; it is not allowed in an attribute_definition_clause.

by:

- The Storage_Pool attribute is not defined for a remote access-to-class-wide type; the expected type for an allocator shall not be a remote access-to-class-wide type. A remote access-to-class-wide type shall not be an actual parameter for a generic formal access type. The Storage_Size attribute of a remote access-to-class-wide type yields 0. The Storage_Pool and Storage_Size aspects shall not be specified for a remote access-to-class-wide type.

Discussion

[Author's note: I noticed when writing this up that the same problem occurs for the Storage_Pool aspect as for the Storage_Size aspect. Not sure how I missed that originally - probably the same way that we missed the entire issue previously. Anyway, I wrote this up assuming that both aspects need to be banned from being specified.]

We definitely don't want these aspects specified by any means, so we adjust the wording to ensure that specifying via an aspect_specification is covered. The existing wording definitely does not cover that, as an attribute is not an aspect, and an attribute_definition_clause is not an aspect_specification.

The text about the Storage_Pool attribute not being defined is still necessary, because we don't want to allow reading the aspect via the attribute as well as not wanting to allow it to be specified.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0215**
WG SECRETARIAT: **Joyce L. Tokar, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2015/07/01**
DEADLINE ON RESPONSE FROM EDITOR: **2015/09/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Jeff Cousins and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:2012 Programming languages — Ada**
QUALIFIER: **Unknown**
REFERENCES IN DOCUMENT: **E.2.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0215 All_Calls_Remote and indirect calls

Working Reference Number AI12-0031-1

Question

Does E.2.3(19/3) apply to indirect calls (that is those through remote access-to-subprogram values)? (Yes.)

Does E.2.3(19/3) apply to dispatching calls (that is those through remote access-to-class-wide types)? (Yes.)

Summary of Response

The All_Calls_Remote aspect applies to all indirect or dispatching remote subprogram calls to the RCI unit as well as to direct calls from outside the declarative region of the RCI unit. Indirect and dispatching remote calls are always considered as being from outside the declarative region and are routed through the PCS.

Corrigendum Wording

Replace E.2.3(19):

> If aspect All_Calls_Remote is True for a given RCI library unit, then the implementation shall route any call to a subprogram of the RCI unit from outside the declarative region of the unit through the Partition Communication Subsystem (PCS); see E.5. Calls to such subprograms from within the declarative region of the unit are defined to be local and shall not go through the PCS.

by:

> If aspect All_Calls_Remote is True for a given RCI library unit, then the implementation shall route any of the following calls through the Partition Communication Subsystem (PCS); see E.5:
>
> - A direct call to a subprogram of the RCI unit from outside the declarative region of the unit;
> - An indirect call through a remote access-to-subprogram value that designates a subprogram of the RCI unit;
> - A dispatching call with a controlling operand designated by a remote access-to-class-wide value whose tag identifies a type declared in the RCI unit.

Discussion

The goal of the All_Calls_Remote aspect is to force calls from outside the declarative region of the RCI unit to go through the PCS. Calls that are local to the RCI unit should always be local and never go through the PCS. This includes indirect and dispatching calls that don't involve remote access types.

This ideal can be implemented easily enough for direct calls, but there are excessive complications for the case of indirect or dispatching calls. In particular, it would be difficult for the implementation to determine if a call was local to the RCI unit, and thus whether the call should go through the PCS.

To avoid these complications, we say that all indirect or dispatching remote subprogram calls to an All_Calls_Remote RCI unit are assumed to be from outside the declarative region of the RCI unit, and therefore go through the PCS.

This has the additional benefit that All_Calls_Remote means all calls are remote from outside the declarative region of the RCI. This eliminates the possibility for requests to rename the pragma to Almost_All_Calls_Remote, or There_Is_A_Possibility_That_The_Call_You_Make_To_This_Package_Could_Be_Remote, etc.